

**PRAIRIE VIEW A&M UNIVERSITY**  
**ROY G. PERRY COLLEGE OF ENGINEERING**  
**DEPARTMENT OF Electrical and Computer ENGINEERING**  
**ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**Spring 2022**

**I. COURSE DESCRIPTION IN ABET FORMAT**

**II. ASSESSMENT RESULTS**

**Program Outcomes Assessed in Course**

**Assessment Summary of Class Semester Averages for Outcomes**

**End of Semester Course Outcomes Assessment Reports**

**III. EXAM1**

**Summary of Topics covered by EXAM1**

(Summary of topics can be a separate page or include on the assignment itself)

**Above Average (Graded original work)**

**Average (Graded original work)**

**Below Average (Graded original work)**

**IV. EXAM2**

**Summary of Topics covered by EXAM2**

**Above Average (Graded original work)**

**Average (Graded original work)**

**Below Average (Graded original work)**

**V. PROJECT I**

**Project I description**

**Above Average (Graded original work)**

**Average (Graded original work)**

**Below Average (Graded original work)**

**VI. FINAL EXAM**

**Summary of Topics covered by FINAL EXAM**

**Above Average (Graded original work)**

**Average (Graded original work)**

**Below Average (Graded original work)**

**VII. PROJECT II**

**Project II description**

**Above Average (Graded original work)**

**Average (Graded original work)**

**Below Average (Graded original work)**

# Course Syllabus in ABET Format

|    |                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----|--------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | <b>Course number and name:</b>                                                                                     | ELEG 4339-P01 Computer Organization and Design                                                                                                                                                                                                                                                                                                                                                                                        |
| 2. | <b>Credits and contact hours:</b>                                                                                  | 3 credit hours; 6 contact hours                                                                                                                                                                                                                                                                                                                                                                                                       |
| 3. | <b>Instructor's or course coordinator's name:</b>                                                                  | Dr. Xiangfang Li; Dr. Suxia Cui                                                                                                                                                                                                                                                                                                                                                                                                       |
| 4. | <b>Text book: Title:</b>                                                                                           | <i>"Computer Organization and Design, The Hardware/Software Interface- RISC-V edition" )</i>                                                                                                                                                                                                                                                                                                                                          |
|    | <b>Author:</b>                                                                                                     | David A. Patterson & John L. Hennessy                                                                                                                                                                                                                                                                                                                                                                                                 |
|    | <b>Publisher, Year &amp; Edition:</b>                                                                              | 2nd edition (December 31, 2020)                                                                                                                                                                                                                                                                                                                                                                                                       |
|    | a. Other Supplemental material                                                                                     | <p>(1)David A. Patterson &amp; John L. Hennessy, "Computer Organization and Design, The Hardware/Software Interface", 5th Edition. Publication Date: October 10, 2013 ISBN-10: 0124077269 ISBN-13: 978-0124077263</p> <p>(2)David A. Patterson &amp; John L. Hennessy, "Computer Organization and Design, The Hardware/Software Interface- ARM edition", 1<sup>st</sup> Edition. 2016 ISBN-13: 978-0128017333 ISBN-10: 0128017333</p> |
| 5. | <b>Specific course information</b>                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|    | a. Brief description of the content of the course (catalog description)                                            | (3-0) Credit 3 semester hours. An introduction to computer organization using assembly and machine language. Number representation, computer arithmetic, instruction sets, I/O interrupts, and programming interrupts. Projects involve detailed study and use of a specific computer hardware and software system.                                                                                                                   |
|    | b. Prerequisites or Co-requisites                                                                                  | ELEG 3063 or ELEG 2311 : Logic Circuits                                                                                                                                                                                                                                                                                                                                                                                               |
|    | c. *Indicate whether a required, elective, or selected elective (as per definition provided) course in the program | Required                                                                                                                                                                                                                                                                                                                                                                                                                              |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                  |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 6. | <p>a. <b>Specific goals for the course</b> specific outcomes of instruction (e.g. The student will be able to explain the significance of current research about a particular topic.)</p> <p><i>Students will be able to :</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | <p>b. <b>Explicitly indicate which of the student outcomes listed in Criterion 3</b> or any other outcomes are addressed by the course.</p> <p><i>The specific goals for the course specific outcomes listed on the left column will address the following student learning outcomes:</i></p>                    |
|    | <p>The objective of this course is to teach students (i) basic concepts of computer organization and hardware, (ii) help them understand the relationship between hardware and software and to focus on the concepts that are the basis for current computers, and (iii) recognize the generation change in computing from the PC era to the PostPC era. Materials covered included but not limited to: Instructions executed by a processor and how to use these instructions in simple assembly language programs; how to design the datapath and control for pipelined and non-pipelined processors; data and control hazards; the principles of memory hierarchy; and how processors, memory, and I/O are combined into a computer.</p> | <p><b>Outcome 7</b></p> <p>An ability to acquire and apply new knowledge as needed, using appropriate learning strategies.</p> <p>(i) Acquire new knowledge using the appropriate learning strategies</p> <p>(ii) Apply new knowledge to solve problems</p>                                                      |
| 7  | <p><b>Brief list of topics to be covered</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | <ul style="list-style-type: none"> <li>• Computer Abstraction and Technology</li> <li>• Instructions: Language of the Computer</li> <li>• Arithmetic for Computers</li> <li>• The Processor</li> <li>• Large and Fast: Exploiting Memory Hierarchy</li> <li>• Parallel Processors for Client to Cloud</li> </ul> |

\***Required** courses are required of all students in the program, **elective** courses (often referred to as open or free electives) are optional for students, and **selected elective** courses are those for which students must take one or more courses from a specified group

## Additional Course Syllabus Information

**COURSE:** ELEG4339 Computer Organization & Design

**CLASS SCHEDULE:** TR (8:00am -9:20am)

**OFFICE** EE building 336

**Office Hours:** T, R, and W 10:00am-1:00pm; or by appointment  
Or by appointment

### GRADING POLICY:

| Course Grade Requirement      | Value       |
|-------------------------------|-------------|
| 1. Attendance                 | 10%         |
| 2. Homework                   | 10%         |
| 3. Quiz(s)                    | 5%          |
| 4. Team Project(s)            | 25%         |
| 5. Exams (two exams)          | 30%         |
| 6. Final Exam (Comprehensive) | 20%         |
| <b>Total:</b>                 | <b>100%</b> |

### Grade Determination

A = 85% or better

B = 75 – 84%

C = 65 – 74%

D = 55 – 64%

F = Less than 54%

Any deviation from the above grading policy will be discussed in advance with the class. Such deviations will be considered only in situations to provide learning enrichment opportunities for the entire class.

### Course Procedures or Additional Instructor Policies

#### ON-LINE RESOURCES:

**Ecourses** (<http://ecourses.pvamu.edu/>) will be used to disseminate course materials and organize discussions about the course to help improve the learning experience of the students.

#### HOMEWORK POLICY:

Homework and design problems will be assigned to complement each unit of instruction. Students will be required to turn in or demonstrate each assignment. Homework has to be handed on time. Late homework will **not** be accepted unless due to acceptable reasons (as defined by University policy). Make up test/quiz only apply to excused absence of class of the test date. Student must contact instructor immediately for scheduling make up time **within one week of the missing test/quiz**.

#### QUIZ/EXAM POLICY:

There will be multiple quizzes/exams throughout this course to motivate students to follow the instruction closely. The quizzes/exams will be administered in the beginning of the class. There will be no make up quizzes/exams if you are absent or late for the class. Make up quizzes/exams only apply to excused absence, student must contact instructor immediately for scheduling make up time **within one week of the missing quizzes/exams**.

#### PROJECT/ASSIGNMENT POLICY:

There will be assigned project(s)/assignment. The project(s) will help the students to absorb the lectures through an interactive approach. Students are required to hand in the project reports.

#### EXAM POLICY:



There are one comprehensive final exam. Every student must take the exam on the assigned dates. Any student who misses an exam without a valid excuse will automatically receive zero for that exam. Make-up exam will be administered in accordance with university policy.

#### **ETHICS:**

Cheating or plagiarism on assignments or exams will not be tolerated. Proven cases of ethical violations will result in a zero for the assignment/exam and possibility of further disciplinary actions in accordance with university policies.

#### **CLASS ATTENDANCE:**

See the University Class Attendance Policy, Effective September 1, 1998. PVAMU requires regular class attendance. Attending all classes supports full academic development of each student. Excessive absenteeism may result in a student's course grade being reduced to a grade of "F". Accumulation of one week of unexcused absences constitutes excessive absenteeism.

#### **COLLEGE OF ENGINEERING TEXTBOOK POLICY**

Students must acquire the textbook that is listed as "required" on the course syllabus. The textbook must be acquired by the 10th class day. Students are not allowed to share textbooks with other students who are currently registered in the same class. Failure to acquire (or show proof of purchase) the required textbook by the 10th class day will result in the student being administratively dropped from the course. The University will assess financial obligations for the course to the student as with any other dropped class according to the fee schedule. In addition, your financial aid may be affected by the subsequent registration action(s). [Go to http://www.pvamu.edu/pages/195.asp](http://www.pvamu.edu/pages/195.asp) for the Roy G. Perry College of Engineering Textbook Policy.

#### **CONDUCT:**

1. Students will conduct themselves in a manner that is respectful to their fellow classmates
2. Cell phones **MUST** be turned off during class time.
3. No cell phones or other electronic devices during exams and tests.
4. Students are **NOT** allowed to wear caps/hats in class

#### **AMERICANS WITH DISABILITY (ADA) STATEMENT:**

Students with disabilities who believe that they may need an academic adjustment in this class, are encouraged to contact the Office of Disability Services at (936) 857-2610/2620 as soon as possible. Once you have received a letter of adjustment from the office, kindly make an appointment with me to discuss appropriate adjustments for this class.

### **Semester Calendar (Tentative Schedule for Topics)**

| <b>WEEK</b>         | <b>Instructor</b> | <b>TOPICS</b>                                                | <b>ASSIGNMENTS</b> |
|---------------------|-------------------|--------------------------------------------------------------|--------------------|
| Week 1: Jan.18, 20  |                   | Introduction, Chap 1 – Computer Abstractions and technology  |                    |
| Week 2: Jan. 25, 27 |                   | Chap 1 – Computer Abstractions and technology                | HW1                |
| Week 3: Feb. 1, 3   |                   | Chap 1 – Computer Abstractions and technology                |                    |
| Week 4: Feb. 8, 10  |                   | Chap 2 – Instructions: Language and computer                 |                    |
| Week 5: Feb. 15, 17 |                   | Chap 2 – Instructions: Language and computer                 | HW2                |
| Week 6: Feb. 22, 24 |                   | Chap 2 – Instructions: Language and computer<br><b>EXAM1</b> |                    |
| Week 7: Mar. 1, 3   |                   | Chap 3 – Arithmetic for computer                             |                    |
| Week 8: Mar.8, 10   |                   | Chap 3 – Arithmetic for computer                             | HW3                |

|                      |  |                                                                                          |     |
|----------------------|--|------------------------------------------------------------------------------------------|-----|
| Week 9: Mar.14-19    |  | Spring Break                                                                             |     |
| Week 10: Mar.22, 24  |  | Chap 3 – Arithmetic for computer                                                         |     |
| Week 11: Mar. 29,31  |  | Chap 4 – The processor                                                                   |     |
| Week 12: Apr. 5, 7   |  | Chap 4 – The processor                                                                   | HW4 |
| Week 13: Apr. 12, 14 |  | Chap 4 – The processor,<br><b>EXAM2</b>                                                  |     |
| Week 14: Apr. 19, 21 |  | Chapter 5: Large and Fast: Exploiting Memory Hierarchy                                   | HW5 |
| Week 15: Apr. 26, 28 |  | Chapter 5: Large and Fast: Exploiting Memory Hierarchy                                   |     |
| Week 16: May 3rd     |  | Chap 6 Parallel processor from client to cloud;<br>Project presentation (if time allows) |     |
| Week 16:             |  | Final Exam Period (May 5-12 <sup>th</sup> )                                              |     |
|                      |  |                                                                                          |     |

## Student Support and Success

### John B. Coleman Library

The John B. Coleman Library's mission is to enhance the scholarly pursuit of knowledge, to foster intellectual curiosity, and to promote life-long learning and research through our innovative services, resources, and cultural programs, which support the Prairie View A&M University's global mission of teaching, service, and research. It maintains library collections and access both on campus, online, and through local agreements to further the educational goals of students and faculty. Website: <https://www.pvamu.edu/library/>; Phone: 936-261-1500

### Academic Advising Services

Academic Advising Services offers students a variety of services that contributes to student success and leads towards graduation. We assist students with understanding university policies and procedures that affect academic progress. We support the early alert program to help students get connected to success early in the semester. We help refer students to the appropriate academic support services when they are unsure of the best resource for their needs. Faculty advisors support some students in their respective colleges. Your faculty advisor can be identified in PantherTracks. Advisors with Academic Advising Services are available to all students. We are located across campus. Find your advisor's location by academic major at [www.pvamu.edu/advising](http://www.pvamu.edu/advising). Phone: 936-261-5911

### The University Tutoring Center

The University Tutoring Center (UTC) offers free tutoring and academic support to all registered PVAMU students. The mission of the UTC is to help provide a solid academic foundation that enables students to become confident, capable, independent learners. Competent and caring staff and peer tutors guide students in identifying, acquiring, and enhancing the knowledge, skills, and attitudes needed to reach their desired goals. Tutoring and academic support are offered face-to-face in the UTC, in virtual face-to-face sessions (<https://www.pvamu.edu/student-success/sass/university-tutoring-center/>), and through online sessions (<https://www.pvamu.edu/pvplace/>). Other support services available for students include Supplemental Instruction, Study Break, Academic Success Workshops, and Algebra Study Jam. Location: J. B. Coleman Library, Rm. 307; Phone: 936-261-1561; Email: [pvtutoring@pvamu.edu](mailto:pvtutoring@pvamu.edu); Website: <https://www.pvamu.edu/student-success/sass/university-tutoring-center/>

### Writing Center

The Writing Center provides well-trained peer tutors to assist students with writing assignments at any stage of the writing process. Tutors help students with various writing tasks from understanding assignments,

brainstorming, drafting, revising, editing, researching, and integrating sources. Students have free access to Grammarly online writing assistance. Grammarly is an automated proofreading and plagiarism detection tool. Students must register for Grammarly by using their student email address. In addition, students have access to face-to-face and virtual tutoring services either asynchronously via email or synchronously via Zoom. Location: J. B. Coleman Library, Rm. 209; Phone: 936-261-3724; Website: <https://www.pvamu.edu/student-success/writing-center/>; Grammarly Registration: <https://www.grammarly.com/enterprise/signup>

### **Academic Early Alert**

Academic Early Alert is a proactive system of communication and collaboration between faculty, academic advisors, and PVAMU students that is designed to support student success by promptly identifying issues and allowing for intervention. Academic Early Alerts help students by providing a central location to schedule advising appointments, view advisor contact information, and request assistance. Students who recognize that they have a problem that is negatively affecting their academic performance or ability to continue school may self-refer an Academic Early Alert. To do so, students will log in to PV Place and click on Academic Early Alert on the left sidebar. Phone: 936-261-5902; Website: <https://www.pvamu.edu/student-success/early-alert/>

### **Student Counseling Services**

The Student Counseling Services unit offers a range of services and programs to assist students in maximizing their potential for success: short-term individual, couples, and group counseling, as well as crisis intervention, outreach, consultation, and referral services. The staff is licensed by the State of Texas and assists students who are dealing with academic skills concerns, situational crises, adjustment problems, and emotional difficulties. Information shared with the staff is treated confidentially and in accordance with Texas State Law. Location: Hobart Taylor, 2<sup>nd</sup> floor; Phone: 936-261-3564; Website: <https://www.pvamu.edu/healthservices/student-counseling-services/>

### **Office of Testing Services**

Testing Services serves to create opportunities by offering a suite of exams that aid in the students' academic and professional success. Currently, we administer entrance (HESI A2), college readiness (TSI assessment), Prior Learning (CLEP, DSST), and proctored exams. Location: Wilhelmina Delco, 3<sup>rd</sup> Floor, Rm. 305; Phone: 936-261-3627; Email: [aetesting@pvamu.edu](mailto:aetesting@pvamu.edu); Website: [www.pvamu.edu/testing](http://www.pvamu.edu/testing)

### **Office of Diagnostic Testing and Disability Services**

The Americans with Disabilities Act (ADA) is a federal anti-discrimination statute that provides comprehensive civil rights protection for persons with disabilities. Among other things, this legislation requires that all students with disabilities be guaranteed a learning environment that provides for reasonable accommodation of their disabilities. If you believe you have a disability requiring an accommodation, contact the Office of Disability Services. As a federally-mandated educational support unit, the Office of Disability Services serves as the repository for confidential disability files for faculty, staff, and students. For persons with a disability, the Office develops individualized ADA letters of request for accommodations. Other services include learning style inventories, awareness workshops, accessibility pathways, webinars, computer laboratory with adapted hard and software, adapted furniture, proctoring non-standardized test administrations, ASL interpreters, ALDs, digital recorders, Livescribe, and a comprehensive referral network across campus and the broader community. Location: Hobart Taylor, Rm. 1D128; Phone: 936-261-3583; Website: <https://www.pvamu.edu/disabilityservices/>

### **Center for Instructional Innovation and Technology Services (CIITS)**

Distance Learning, also referred to as Distance Education, is the employment of alternative instructional delivery methods to extend programs and services to persons unable to attend college in the traditional manner. The Center for Instructional Innovation and Technology Services (CIITS) supports student learning through online, hybrid, web-assist, and 2-way video course delivery. For more details and contact information, visit:

<https://www.pvamu.edu/dlearning/distance-learning-2-2/students-2/>; Phone: 936-261-3283

### **Veteran Affairs**

Veterans Services works with student veterans, current military and military dependents to support their transition to the college environment and continued persistence to graduation. The Office coordinates and certifies benefits for both the G.I. Bill and the Texas Hazlewood Act. Location: Evans Hall, Rm. 102; Phone: 936-261-3563; Website: <https://www.pvamu.edu/sa/departments/veteranaffairs/>

### **Office for Student Engagement**

The Office for Student Engagement delivers comprehensive programs and services designed to meet the co-curricular needs of students. The Office implements inclusive and accessible programs and services that enhance student development through exposure to and participation in diverse and relevant social, cultural, intellectual, recreational, community service, leadership development, and campus governance. Location: Memorial Student Center, Rm. 221; Phone: 936-261-1340; Website: <https://www.pvamu.edu/studentengagement/>

### **Career Services**

Career Services supports students through professional development, career readiness, and placement and employment assistance. The Office provides one-on-one career coaching, interview preparation, resume and letter writing, and career exploration workshops and seminars. Services are provided for students at the Northwest Houston Center and College of Nursing in the Medical Center twice a month or on a requested basis. Distance Learning students are encouraged to visit the Career Services website for information regarding services provided. Location: Anderson Hall, 2<sup>nd</sup> floor; Phone: 936-261-3570; Website: <https://www.pvamu.edu/careerservices/>

## **University Rules and Procedures**

### **Academic Misconduct**

Academic dishonesty is defined as any form of cheating or dishonesty that has the effect or intent of interfering with any academic exercise or fair evaluation of a student's performance. The college faculty can provide additional information, particularly related to a specific course, laboratory, or assignment.

You are expected to practice academic honesty in every aspect of this course and all other courses. Make sure you are familiar with the *University Administrative Guidelines on Academic Integrity*, which can be found on the [Academic Integrity webpage](#). Students who engage in academic misconduct are subject to university disciplinary procedures. As listed in the *University Administrative Guidelines on Academic Integrity*, the University Online Catalog, and the Student Code of Conduct, the following are examples of prohibited conduct. This list is not designed to be all-inclusive or exhaustive. In addition to academic sanctions, any student found to have committed academic misconduct that is also a violation of criminal law may also be subject to disciplinary review and action by the Office of Student Conduct (as outlined in the Student Code of Conduct).

### **Forms of Academic Dishonesty:**

1. **Cheating**: Deception in which a student misrepresents that he/she has mastered information on an academic exercise that he/she has not learned, giving or receiving aid unauthorized by the instructor on assignments or examinations. Examples: unauthorized use of notes for a test; using a "cheat sheet" on a quiz or exam; any alteration made on a graded test or exam which is then resubmitted to the teacher;
2. **Plagiarism**: Careless or deliberate use of the work or the ideas of another; representation of another's work, words, ideas, or data as your own without permission or appropriate acknowledgment. Examples: copying another's paper or answers, failure to identify information or essays from the internet and submitting or representing it as your own; submitting an assignment which has been partially or wholly done by another and claiming it as yours; not properly acknowledging a source which has been summarized or paraphrased in your work; failure to acknowledge the use of another's words with quotation marks;
3. **Collusion**: When more than one student or person contributes to a piece of work that is submitted as

the work of an individual;

4. Conspiracy: Agreeing with one or more persons to commit an act of academic/scholastic dishonesty; and
5. Multiple Submission: Submission of work from one course to satisfy a requirement in another course without explicit permission. Example: using a paper prepared and graded for credit in one course to fulfill a requirement and receive credit in a different course.

### **Nonacademic Misconduct**

The university respects the rights of instructors to teach and students to learn. Maintenance of these rights requires campus conditions that do not impede their exercise. Campus behavior that interferes with either (1) the instructor's ability to conduct the class, (2) the inability of other students to profit from the instructional program, or (3) campus behavior that interferes with the rights of others will not be tolerated. An individual engaging in such disruptive behavior may be subject to disciplinary action. The Office of Student Conduct will adjudicate such incidents under nonacademic procedures.

### **Sexual Misconduct**

Sexual harassment of students and employees at Prairie View A&M University is unacceptable and will not be tolerated. Any member of the university community violating the university's sexual harassment policy will be subject to disciplinary action. In accordance with the Texas A&M University System guidelines, your instructor is obligated to report to the Office of Title IX Compliance ([titleixteam@pvamu.edu](mailto:titleixteam@pvamu.edu)) any instance of sexual misconduct involving a student, which includes sexual assault, stalking, dating violence, domestic violence, and sexual harassment, about which the instructor becomes aware during this course through writing, discussion, or personal disclosure. The faculty and staff of PVAMU actively strive to provide a learning, working, and living environment that promotes respect that is free from sexual misconduct, discrimination, and all forms of violence. If students, faculty, or staff would like assistance or have questions, they may contact the Title IX Coordinator at 936-261-2144 or [titleixteam@pvamu.edu](mailto:titleixteam@pvamu.edu). More information can be found at [www.pvamu.edu/titleix](http://www.pvamu.edu/titleix), including confidential resources available on campus.

### **Pregnancy, Pregnancy-related, and Parenting Accommodations**

Title IX of the Education Amendments of 1972 prohibits sex discrimination, which includes discrimination based on pregnancy, marital status, or parental status. Students seeking accommodations related to pregnancy, pregnancy-related conditions, or parenting (reasonably immediate postpartum period) are encouraged to contact Student Disability Services or the Dean of Students' Office for additional information and to request accommodations.

### **Non-Discrimination Statement**

Prairie View A&M University does not discriminate on the basis of race, color, sex, religion, national origin, age, disability, genetic information, veteran status, sexual orientation, or gender identity in its programs and activities. The University is committed to supporting students and complying with The Texas A&M University System non-discrimination policy. It seeks to establish an environment that is free of bias, discrimination, and harassment. If you experience an incident of discrimination or harassment, we encourage you to report it. If you would like to speak with someone who may be able to afford you privacy or confidentiality, there are individuals who can meet with you. The Director of Equal Opportunity & Diversity has been designated to handle inquiries regarding the non-discrimination policies and can be reached at Harrington Science Building, Suite 109 or by phone at 936-261-1744 or 1792.

### **Class Attendance Policy (See the University Online Catalog for Full Attendance Policy)**

Prairie View A&M University requires regular class attendance. Attending all classes supports the full academic development of each learner, whether classes are taught with the instructor physically present or via distance learning technologies such as interactive video and/or the internet. Excessive absenteeism, whether excused or unexcused, may result in a student's course grade being reduced or in the assignment of a grade of "F." Absences are accumulated beginning with the first day of class during regular semesters and summer terms. Each faculty member will include the University's attendance policy in each course syllabus.

## **Student Academic Appeals Process**

Authority and responsibility for assigning grades to students rest with the faculty. However, in those instances where students believe that miscommunication, errors, or unfairness of any kind may have adversely affected the instructor's assessment of their academic performance, the student has a right to appeal by the procedure listed in the University Online Catalog and by doing so within thirty days of receiving the grade or experiencing any other problematic academic event that prompted the complaint.

## **Technical Considerations**

### **Minimum Recommended Hardware and Software:**

- Intel PC or Laptop with Windows 10 or later version; Mac with OS High Sierra\*
- Smartphone or iPad/Tablet with Wi-Fi\*
- High-speed Internet access
- 8 GB Memory
- Hard drive with 320 GB storage space
- 15" monitor, 800x600, color or 16 bit
- Sound card w/speakers
- Microphone and recording software
- Keyboard & mouse
- Most current version of Google Chrome, Safari, or Firefox

**Note:** Be sure to enable Java & pop-ups in the Web browser preferences

\* Smartphones, Google Chrome books, and Android tablets may not be supported. iPads are the only tablets supported.

### **Participants should have a basic proficiency of the following computer skills:**

- Sending and receiving email
- A working knowledge of the Internet
- Microsoft Word (or a program convertible to Word)
- Acrobat PDF Reader
- Windows or Mac OS
- Video conferencing software

### **Netiquette (online etiquette)**

Students are expected to participate in all discussions and virtual classroom chats as directed. Students are to be respectful and courteous to others on discussion boards. Foul or abusive language will not be tolerated. Do not use ALL CAPS for communicating to others AS IT CAN BE INTERPRETED AS YELLING. Avoid slang terms such as "wassup?" and texting abbreviations such as "u" instead of "you." Limit and possibly avoid the use of emoticons. Be cautious when using humor or sarcasm as tone is sometimes lost in an email or discussion post, and the message might be taken seriously or sound offensive.

### **Video Conferencing Etiquette**

When using Zoom, WebEx, or other video conferencing tools, confirm the visible area is tidy, clear of background clutter, inappropriate or offensive posters, and other distractions. Ensure you dress appropriately and avoid using high traffic or noisy areas. Stay muted when you are not speaking and avoid eating/drinking during the session. Before the class session begins, test audio, video, and lighting to alleviate technology issues.

### **Technical Support**

Students should go to <https://mypassword.pvamu.edu/> if they have password issues. The page will provide instructions for resetting passwords and contact information if login issues persist. For other technical questions regarding eCourses, call the Center for Instructional Innovation and Technology Services at 936-261-3283 or email ciits@pvamu.edu.

## **Communication Expectations and Standards**



Emails or discussion postings will receive a response from the instructor, usually in less than 48 hours. Urgent emails should be marked as such. Check regularly for responses.

### **Discussion Requirement**

Online courses often require minimal to no face-to-face meetings. However, conversations about the readings, lectures, materials, and other aspects of the course can occur in a seminar fashion. The use of the discussion board will accomplish this. The instructor will determine the exact use of discussion boards.

**It is strongly suggested** that students type their discussion postings in a word processing application such as Word and save it to their PC or a removable drive before posting to the discussion board. This is important for two reasons: 1) If for some reason your discussion responses are lost in your online course, you will have another copy; 2) Grammatical errors can be greatly minimized by the use of the spell-and-grammar check functions in word processing applications. Once the post(s) have been typed and corrected in the word processing application, copy and paste to the discussion board.

### **COVID-19 Campus Safety Measures [NOTE: Delete this section when the COVID-19 pandemic is over]**

To promote public safety and protect students, faculty, and staff during the coronavirus pandemic, PVAMU has adopted policies and practices to limit virus transmission.

- **Self-monitoring** - Students should follow CDC recommendations for self-monitoring. Students who have a fever or exhibit symptoms of COVID-19 should participate in class remotely and should not participate in face-to-face instruction.
- **Face Coverings** - Face coverings (cloth face covering, surgical mask, etc.) are recommended in classrooms, teaching laboratories, common spaces such as lobbies and hallways, public study spaces, libraries, academic resource, and support offices, and outdoor spaces where 6 feet of physical distancing is challenging to maintain reliably.
- **Physical Distancing** - Physical distancing should be maintained between students, instructors, and others in course and course-related activities where possible.

- **Personal Illness and Quarantine** - Students required to quarantine are to participate in courses and course-related activities remotely and must not attend face-to-face course activities. Students should notify their instructors of the quarantine requirement. Students under quarantine are expected to participate in courses and complete graded work unless they have symptoms that are too severe to participate in course activities. Students experiencing personal injury or illness that is too severe for the student to attend class qualify for an excused absence. To receive an excused absence, students must provide appropriate documentation to the Office for Student Conduct, [studentconduct@pvamu.edu](mailto:studentconduct@pvamu.edu).

# End of Semester Course Assessment Report

This summary contains the following Information

## 1. Performance Statistics

- Total number of students in class
- The class average performance (**stated as %**) in each course outcome assessed in this class and the **two previous times course was assessed**.
- The acceptable class average for the class in each outcome area as set by your Department. **You cannot use any number of your choice**
- Percentage of students who scored below the expected average in each outcome.

## 2. Implementation Summary

- Brief summary of plans implemented during the semester based on last assessment report

## 3. Perceived Problems

- Instructor's **critical evaluation** of perceived problems that affected students' performance. **Realistic problems should be identified throughout the semester and summarized in this section of the report.**

## 4. New Plans for Addressing Problems

- Instructor's plans for addressing the perceived problems the next time the class is taught. **These should be specific enough for another instructor to implement. Most of these should be things that, you, the instructor can implement to help students to improve. Be creative.**

## 5. Overall Trend over last three assessment periods

- Instructors summary of overall performance trend for each outcome measured over the last three periods.

## 5. Were Expectations Met?

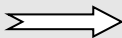
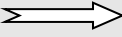
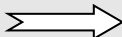
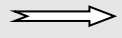
Answer **YES** if the **class average was above the expected average** and the **stated percentage of students** scored at or above the



**Table 2.1 END OF SEMESTER COURSE OUTCOME ASSESSMENT REPORT**  
**ELEG4339-P01 COMPUTER ORGANIZATION & DESIGN**  
**SPRING 2021 SEMESTER**

**Report Prepared by: Drs. Xiangfang Li, Suxia Cui**

**Report Date: MAY 17, 2022**

| Semester                             | Analysis Type                                                                       |    | Number of Students                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Outcome 7                                                                                                                                                                                                            |               |                                   |
|--------------------------------------|-------------------------------------------------------------------------------------|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-----------------------------------|
|                                      |                                                                                     |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | An ability to acquire and apply new knowledge as needed, using appropriate learning strategies.<br>(i) Acquire new knowledge using the appropriate learning strategies<br>(ii) Apply new knowledge to solve problems |               |                                   |
|                                      |                                                                                     |    | Expected Average                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | % Students Above Expected Average                                                                                                                                                                                    | Class Average | % Students Above Expected Average |
| Spring 2022                          | Direct                                                                              | 12 | 75                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 70%                                                                                                                                                                                                                  | 80.1          | 91.6%                             |
| Spring 2021                          | Direct                                                                              | 32 | 75                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 70%                                                                                                                                                                                                                  | 84.8          | 90%                               |
| Spring 2020                          | Direct                                                                              | 7  | 75                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 70%                                                                                                                                                                                                                  | 83            | 85%                               |
| <b>Implementation Summary</b>        |    |    | In order to intrigue and involve students better, two projects are given this semester. The first project is based on Venus simulators, to ask the students get familiar with the simulator and learn RISC-V assembly language programming. The second project is about modern processor design. The students are asked to explore some modern processor based on the material learned in the class, and other materials from online resources, technical papers, etc. A report is collected and a mini-conference is organized near the end of the semester to presents their findings. It is hard to separate and allocate each project to each sub-outcome. The combined score is used to evaluate the outcome.                                                                                                                                    |                                                                                                                                                                                                                      |               |                                   |
| <b>Perceived Problems</b>            |   |    | Comparing to 2021, face to face delivery of the course materials are adopted in this semester. It makes the feedback and communication much clearer. It is much easier to adjust the pace of the course based on students' understanding. In general, the teaching effect is much better. This course of "computer organization and design" includes many very modern and timely topics. In each semester, the time limitation is clearly observed, it is not easy to include every aspect within the class time frame. The instructor always wish there are more time to cover more topics, and the content itself is very broad, rich, and keep evolving. Hence, how to encourage the students to explore the new knowledge independently based on the essential foundation established in class is very critical and challenging at the same time. |                                                                                                                                                                                                                      |               |                                   |
| <b>Plans for Addressing Problems</b> |  |    | The two projects solutions adopted this semester is very promising. The students practice the assembly language programing in project 1, and they actively learning the modern processor design in project 2. The effect of mini-conference is also very good. Not only students need to present their own findings, they also could learn from other teams' presentation. All the processor case study are closely related to the what is happening/trendy in industry. Similar methods will be adopted in the coming years, and each year, the project topic should be keep evolving. Such project-based approach would promote students' active learning.                                                                                                                                                                                          |                                                                                                                                                                                                                      |               |                                   |
| <b>Overall Trend over Periods</b>    |  |    | Overall trend is improving. Students' performance is satisfactory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                      |               |                                   |
| <b>Were Expectations Met?</b>        | Average ≥ 70%? and students avg. ≥ 70%?                                             |    | <b>YES</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                      |               |                                   |

Grade Sheet Showing Student and Class Performance in Course Outcomes  
**ELEG 4339 Computer Organization and Design**  
**SPRING 2022 Semester**

Scores for ABET Outcome 7 (Total 12 students)

| Record         | Student Name | Project 1 (10%) | Project 2 (15%)   |              | Score (outcome 7) | Met or Not (>75)      |
|----------------|--------------|-----------------|-------------------|--------------|-------------------|-----------------------|
|                |              |                 | Presentation (5%) | Report (10%) | Weighted Average  |                       |
| 1              |              | 93              | 83                | 80           | 85.8              | Y                     |
| 2              |              | 65              | 80                | 85           | 76                | Y                     |
| 3              |              | 85              | 80                | 85           | 84                | Y                     |
| 4              |              | 83              | 85                | 80           | 82.2              | Y                     |
| 5              |              | 80              | 88                | 75           | 79.6              | Y                     |
| 6              |              | 75              | 80                | 75           | 76                | Y                     |
| 7              |              | 75              | 88                | 85           | 81.6              | Y                     |
| 8              |              | 93              | 83                | 80           | 85.8              | Y                     |
| 9              |              | 40              | 80                | 85           | 66                | N                     |
| 10             |              | 83              | 85                | 80           | 82.2              | Y                     |
| 11             |              | 85              | 85                | 80           | 83                | Y                     |
| 12             |              | 75              | 85                | 80           | 79                | Y                     |
| <b>AVERAGE</b> |              | <b>78</b>       | <b>83.5</b>       | <b>80.8</b>  | <b>80.1</b>       | <b>91.6 % over 75</b> |

# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**SPRING 2022**

**Title of Assignment: EXAM1**

## **TOPICS COVERED IN EXAM1**

**Exam1 covered the following topics:**

1. Computer abstractions and technology
2. Computer Performance, define computer performance in terms of CPU time
3. CPI: Clock cycles per instruction
4. Amdahl's Law
5. Instructions: Language of the computer
6. RISC-V assembly language
7. Operations of computer hardware, operands, different instructions representation in computer

# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**Exam1**

**ABOVE AVERAGE**

## **Above Average Student Original Graded Assignment**

**ELEG 4339: Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Exam 1

Spring 2022

Name: (Print) John Barrett

(Sign) John W. Barrett

|              |            |              |
|--------------|------------|--------------|
| 1            | 16         | 16           |
| 2            | 42         | 39.5         |
| 3            | 12         | 12           |
| 4            | 10         | 10           |
| 5            | 10         | 10           |
| 6            | 12         | 10           |
| 7            | 8          | 4            |
|              |            |              |
| <b>Total</b> | <b>110</b> | <b>101.5</b> |

Instructions:

You have 80 minutes to complete this exam. This exam counts for 15% of your course grade.



(16pts) 1. Short answer (2 points each)

(1.1). Computer architects divide processors based on design criteria of instruction sets into two broad categories: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Choose **all** the following characteristics that are **applicable to RISC processor**.

- ☐ a. Variable-length instructions
- ☒ b. Fixed-length instructions
- ☐ c. Many instructions (typically, hundreds)
- ☒ d. Strives for a minimum instruction set
- ☒ e. Each instruction performs a basic computation
- ☐ f. Each instruction can perform an arbitrarily complex computation

(1.2). Do the following changes to a computer system increase throughput, decrease response time, or both?

Replacing the processor in a computer with a faster version.

*Both*

(1.3) Please describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

*A compiler translates the code typically into Assembly. Then the Assembler translates the Assembly into machine language.*

(1.4). Representing  $-5_{10}$  in 8-bit binary number using 2's complement

$$5_{10} \Rightarrow 101_2 \Rightarrow \begin{array}{r} 11111010 \\ 11110111 \\ \hline 11110111 \end{array} \Rightarrow -5_{10}$$

(1.5) Representing  $-5_{10}$  in 32-bit binary number using 2's complement

*1111, 1111, 1111, 1111, 1111, 1111, 1111, 1011*

(1.6) Which are five classic components of a computer?

*Input, output, memory, Data path, control*

(1.7) For a given function, which programming language likely takes more lines of code?

- (a) C      (b) RISC-V Assembly language

(1.8) What is ISA? Please give one example here.

*Instruction Set Architecture  
Risc-V*

(42 pts) 2. Medium answer (6 points each)

(2.1). A program requires executing 100 million instructions on a processor which typically with CPI of 2 and a 2GHz clock. How much time will the program take?

$T_{CPU} = \frac{(100 \text{ M}) (2) \text{ CPI}}{(2 \text{ GHz})} = 100 \text{ seconds}$

(2.2). Assuming variable  $a$  is stored in register  $x5$ ,  $b$  is stored in  $x6$ ,  $c$  is stored in  $x7$ , and the result  $h$  is to be stored in  $x8$ , what is the RISC-V assembly instructions equivalent to the C statement

$$h = (a+b) - (c-5)$$

Add  $x9, x5, x6$

~~Sub  $x10, x9, x7$~~  Addi  $x10, x7, -5$

Sub  $x8, x9, x10$

(2.3). A given application written in Java runs 40 seconds on a desktop processor. A new Java compiler is released that requires only 0.7 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1 time.

How long can we expect the application to run using the new compiler?

$$40 \cdot 0.7 \cdot 1.1 = 30.8 \text{ seconds}$$

$$T_{CPU_1} = 40 = \frac{(\# \text{ Inst}) (CPI)}{Freq} \quad T_{CPU_2} = \frac{(\# \text{ Inst}) (CPI) (0.7) (1.1)}{Freq}$$

(2.4). Translate the c statement  $A[10] = h + A[4]$  to RISC-V assembly instructions, assuming  $h$  in  $x10$ , base address of  $A$  in  $x18$ , and each array element is a word (32 bit) (RV-32 is assumed here).

lw  $x6, 16(x18)$

Add  $x9, x10, x6$

sw  $x9, 40(x18)$

- 6 (2.5). Suppose a program runs in 100 seconds on a computer, with **multiply operations** account for 70 seconds of the time. How much improvement in **multiply operations'** performance to get the program to run 2 times faster?

$$100 = M + O \Rightarrow 100 = 70 + 30$$

$$50 = \frac{70}{n} + 30 \Rightarrow 20 = \frac{70}{n} \Rightarrow n = \frac{70}{20} = 3.5$$

*You need a 3.5 times improvement*

- 6 (2.6). When determine the CPU performance, CPU time is related to Instruction count, CPI and clock rate. Please fill out the following table to check which aspects are affected by the specific hardware and software component.

|                      | Instruction Count | CPI | Clock Rate |
|----------------------|-------------------|-----|------------|
| Algorithm            | ✓                 | ✓   |            |
| ISA                  | ✓                 | ✓   | ✓          |
| Compiler             | ✓                 | ✓   |            |
| Programming language | ✓                 | ✓   |            |

- 5.5 (2.7). Given a 32-bit binary number, please demonstrate **the range** it could represent in 2 different cases.

(a) case 1: unsigned integer

$$[0, 4294967296]$$

$$[0, 2^{32} - 1]$$

(b) case 2: signed integer

~~$$[-4294967296, 4294967296]$$~~

$$[-2147483648, 2147483647]$$

12 (12pts) 3. Consider two different implementation of the same instruction set architecture. The instruction can be divided into four classes according to their CPI (class A, B, C, and D), Implementation P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 4. Implementation P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 40% class A; 30% class B; 20% class C; 10% class D

- (a) What is the global CPI for each implementation?  
(b) Which implementation is faster: P1 or P2? By how much?

a) 
$$\#Cycles_1 = (1E6)(0.4)(1) + (1E6)(0.3)(2) + (1E6)(0.2)(3) + (1E6)(0.1)(4) = 2E6$$
  

$$\#Cycles_1 \quad CPI_1 = \frac{\#Cycles_1}{\#of\ Instructions} = \frac{2E6}{1E6} = 2 \quad \boxed{CPI_1 = 2}$$
  

$$\#Cycles_2 = 2E6 \quad CPI_2 = \frac{2E6}{1E6} = 2 \quad \boxed{CPI_2 = 2}$$
  
 b) 
$$T_{CPU1} = \frac{(1E6)(2)}{(2.5E6)} \quad T_{CPU2} = \frac{(1E6)(2)}{(3E6)}$$
  

$$\frac{T_{CPU1}}{T_{CPU2}} = \frac{3E6}{2.5E6} = 1.2$$
  
 b) 
$$\boxed{P_2 \text{ is faster by } 1.2 \text{ times}}$$

10 (10 pts) 4. Please provide the instruction type (2pts), binary (6pts), and hexadecimal (2pts) representation of the following instruction, the table below is given for your reference.

| Instruction          | Format | funct7     | rs2 | rs1 | funct3 | rd         | opcode  |
|----------------------|--------|------------|-----|-----|--------|------------|---------|
| add (add)            | R      | 0000000    | reg | reg | 000    | reg        | 0110011 |
| sub (sub)            | R      | 0100000    | reg | reg | 000    | reg        | 0110011 |
| Instruction          | Format | immediate  |     | rs1 | funct3 | rd         | opcode  |
| addi (add immediate) | I      | constant   |     | reg | 000    | reg        | 0010011 |
| lw (load word)       | I      | address    |     | reg | 010    | reg        | 0000011 |
| Instruction          | Format | immed-iate | rs2 | rs1 | funct3 | immed-iate | opcode  |
| sw (store word)      | S      | address    | reg | reg | 010    | address    | 0100011 |

Lw x3, 100 (x22)

$$\boxed{00001100100, 10110, 010, 00011, 0000011} \quad \text{(Binary)}$$

I format

$$22_{10} = 10110$$

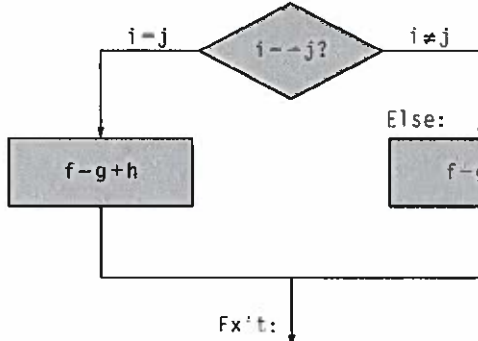
90

$$\boxed{064B2183} \quad \text{(HEX)}$$

(10 pts) 5. Please change the following c statement to RISC-V statements

if (i==j) {f = g+h} else {f = g-h};

Assume the register allocations are: i: x20; j: x21; f: x22; g: x23; h: x24



~~beg x20, x21, HERE~~  
~~Sub x22, x23, x24~~  
~~beg x20, x21, HERE~~  
~~HERE: Add x22, x23, x24~~  
~~EXIT:~~

✓ beg x20, x21, HERE  
 ✓ Sub x22, x23, x24  
 ✓ beg x0, x0, EXIT  
 ✓ HERE: Add x22, x23, x24  
 EXIT:

(12 pts) 6. Please change the following c statement to RISC-V statements.

Assume that base address of array A is in register x10, base address of array B is in register x11, (assuming each array element is a doubleword (64 bit), each register is also 64-bit wide). Variables register allocation are: c: x6, i: x7, and j: x8.

What is the RISC-V assembly code for the C statement?

c = A[i] - B[j]

Slli x12, x7, 3  
 Add x10, x10, x12  
 Slli x13, x8, 3  
 Add x11, x11, x13  
~~Sub x6, x10, x11~~  
 ld x14, 0(x10)  
 ld x15, 0(x11)  
 Sub x6, x14, x15

Register Allocation

(8 pts) 7. Suppose the program counter (PC) is set to 0x 4000 0000

(4 pts)(a) What range of addresses can be reached using RISC-V jump-and-link (jal) instruction?

$[-2097152, 2097150]_{10} \Rightarrow [FFFE0000, 001FFFFE]_{16}$  *this is not final answer*  
 $[13FE000, 401FFFFE]_{16}$  *Final answer*

(4 pts)(b) What range of addresses can be reached using RISC-V branch if equal (beq) instruction?

~~$2^{23} = 8192 \times 2 = 16384$~~   
 ~~$-2 = 8191 \times 2 = 16382$~~   
 $2^{22} = 4096 \times 2 = 8192$   
 $-1 = 4095 \times 2 = 8190$   
 $[-8192, 8190]_{10} \Rightarrow [FFFE0000, 00001FFF]_{16}$   
 $[3FFFE000, 40601FFE]_{16}$



### RISC-V assembly language

| Category      | Instruction                      | Example           | Meaning                            | Comments                                   |
|---------------|----------------------------------|-------------------|------------------------------------|--------------------------------------------|
| Arithmetic    | Add                              | add x5, x6, x7    | $x5 = x6 + x7$                     | Three register operands                    |
|               | Subtract                         | sub x5, x6, x7    | $x5 = x6 - x7$                     | Three register operands                    |
| Data transfer | Add immediate                    | addi x5, x6, 20   | $x5 = x6 + 20$                     | Used to add constants                      |
|               | Load doubleword                  | ld x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Doubleword from memory to register         |
|               | Store doubleword                 | sd x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Doubleword from register to memory         |
|               | Load word                        | lw x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Word from memory to register               |
|               | Load word, unsigned              | lwu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Unsigned word from memory to register      |
|               | Store word                       | sw x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Word from register to memory               |
|               | Load halfword                    | lh x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Halfword from memory to register           |
|               | Load halfword, unsigned          | lhu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Unsigned halfword from memory to register  |
|               | Store halfword                   | sh x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Halfword from register to memory           |
|               | Load byte                        | lb x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Byte from memory to register               |
|               | Load byte, unsigned              | lbu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Byte halfword from memory to register      |
|               | Store byte                       | sb x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Byte from register to memory               |
|               | Load reserved                    | lr.d x5, (x6)     | $x5 = \text{Memory}[x6]$           | Load; 1st half of atomic swap              |
|               | Store conditional                | sc.d x7, x5, (x6) | $\text{Memory}[x6] = x5; x7 = 0/1$ | Store; 2nd half of atomic swap             |
|               | Load upper immediate             | lui x5, 0x12345   | $x5 = 0x12345000$                  | Loads 20-bit constant shifted left 12 bits |
| Logical       | And                              | and x5, x6, x7    | $x5 = x6 \& x7$                    | Three reg. operands; bit-by-bit AND        |
|               | Inclusive or                     | or x5, x6, x8     | $x5 = x6   x8$                     | Three reg. operands; bit-by-bit OR         |
|               | Exclusive or                     | xor x5, x6, x9    | $x5 = x6 \wedge x9$                | Three reg. operands; bit-by-bit XOR        |
|               | And immediate                    | andi x5, x6, 20   | $x5 = x6 \& 20$                    | Bit-by-bit AND reg. with constant          |
|               | Inclusive or immediate           | ori x5, x6, 20    | $x5 = x6   20$                     | Bit-by-bit OR reg. with constant           |
| Shift         | Exclusive or immediate           | xori x5, x6, 20   | $x5 = x6 \wedge 20$                | Bit-by-bit XOR reg. with constant          |
|               | Shift left logical               | sll x5, x6, x7    | $x5 = x6 \ll x7$                   | Shift left by register                     |
|               | Shift right logical              | srl x5, x6, x7    | $x5 = x6 \gg x7$                   | Shift right by register                    |
|               | Shift right arithmetic           | sra x5, x6, x7    | $x5 = x6 \gg x7$                   | Arithmetic shift right by register         |
|               | Shift left logical immediate     | slli x5, x6, 3    | $x5 = x6 \ll 3$                    | Shift left by immediate                    |
|               | Shift right logical immediate    | srlr x5, x6, 3    | $x5 = x6 \gg 3$                    | Shift right by immediate                   |
|               | Shift right arithmetic immediate | sral x5, x6, 3    | $x5 = x6 \gg 3$                    | Arithmetic shift right by immediate        |

|                      |                                   |                  |                            |                                                  |
|----------------------|-----------------------------------|------------------|----------------------------|--------------------------------------------------|
| Conditional branch   | Branch if equal                   | beq x5, x6, 100  | if (x5 == x6) go to PC+100 | PC-relative branch if registers equal            |
|                      | Branch if not equal               | bne x5, x6, 100  | if (x5 != x6) go to PC+100 | PC-relative branch if registers not equal        |
|                      | Branch if less than               | blt x5, x6, 100  | if (x5 < x6) go to PC+100  | PC-relative branch if registers less             |
|                      | Branch if greater or equal        | bge x5, x6, 100  | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
|                      | Branch if less, unsigned          | bltu x5, x6, 100 | if (x5 < x6) go to PC+100  | PC-relative branch if registers less             |
|                      | Branch if greater/equal, unsigned | bgeu x5, x6, 100 | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
| Unconditional branch | Jump and link                     | jal x1, 100      | x1 = PC+4; go to PC+100    | PC-relative procedure call                       |
|                      | Jump and link register            | jalr x1, 100(x5) | x1 = PC+4; go to x5+100    | Procedure return; indirect call                  |

| Name<br>(Field Size) | Field                       | Field  | Field  | Field  | Field         | Field  | Comments                      |
|----------------------|-----------------------------|--------|--------|--------|---------------|--------|-------------------------------|
|                      | 7 bits                      | 5 bits | 5 bits | 3 bits | 5 bits        | 7 bits |                               |
| R-type               | funct7                      | rs2    | rs1    | funct3 | rd            | opcode | Arithmetic instruction format |
| I-type               | immediate[11:0]             |        | rs1    | funct3 | rd            | opcode | Loads & immediate arithmetic  |
| S-type               | immed[11:5]                 | rs2    | rs1    | funct3 | immed[4:0]    | opcode | Stores                        |
| SB-type              | immed[12,10:5]              | rs2    | rs1    | funct3 | immed[4:1,11] | opcode | Conditional branch format     |
| UJ-type              | immediate[20,10:1,11,19:12] |        |        |        | rd            | opcode | Unconditional jump format     |
| U-type               | immediate[31:12]            |        |        |        | rd            | opcode | Upper immediate format        |

Information page

**AVERAGE**

## **Average Student Original Graded Assignment**



**ELEG 4339: Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Exam 1

Spring 2022

Name: (Print) Nasir Pak  
(Sign) Nasir Pak

|              |            |           |
|--------------|------------|-----------|
| 1            | 16         | 15        |
| 2            | 42         | 41.5      |
| 3            | 12         | 9         |
| 4            | 10         | 2         |
| 5            | 10         | 10        |
| 6            | 12         | 10        |
| 7            | 8          | 0.5       |
|              |            |           |
| <b>Total</b> | <b>110</b> | <b>88</b> |

**Instructions:**

You have 80 minutes to complete this exam. This exam counts for 15% of your course grade.

(16pts) 1. Short answer (2 points each)

(1.1). Computer architects divide processors based on design criteria of instruction sets into two broad categories: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Choose **all** the following characteristics that **are applicable to RISC processor**.

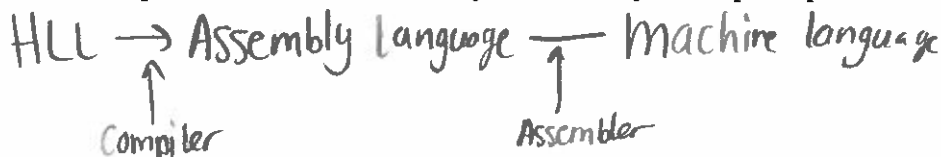
- a. Variable-length instructions
- b. Fixed-length instructions
- c. Many instructions (typically, hundreds)
- d. Strives for a minimum instruction set
- e. Each instruction performs a basic computation
- f. Each instruction can perform an arbitrarily complex computation

(1.2). Do the following changes to a computer system increase throughput, decrease response time, or both?

Replacing the processor in a computer with a faster version.

*It increases throughput and response time*

(1.3) Please describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.



(1.4). Representing  $-5_{10}$  in 8-bit binary number using 2's complement



(1.5) Representing  $-5_{10}$  in 32-bit binary number using 2's complement



(1.6) Which are five classic components of a computer?

*input, output, memory, datapath, control*

(1.7) For a given function, which programming language likely takes more lines of code?

- (a) C      (b) RISC-V Assembly language

(1.8) What is ISA? Please give one example here.

*Instruction set architecture and one example is how it affects how the processor is set up as it affects the instruction count, clock rate, and execution time*

100,000,000

1,000,000

4.5

(42 pts) 2. Medium answer (6 points each)

(2.1). A program requires executing 100 million instructions on a processor which typically with CPI of 2 and a 2GHz clock. How much time will the program take?

$$IC = 1 \times 10^8$$

$$CPI = 2$$

$$CR = 2 \text{ GHz}$$

$$\text{time} = \frac{IC * CPI}{CR} = \frac{(1 \times 10^8) * (2)}{2 \times 10^9} =$$

$$\text{time} = 0.1 \text{ sec}$$

(2.2). Assuming variable  $a$  is stored in register  $x5$ ,  $b$  is stored in  $x6$ ,  $c$  is stored in  $x7$ , and the result  $h$  is to be stored in  $x8$ , what is the RISC-V assembly instructions equivalent to the C statement

$$h = (a+b) - (c-5)$$

$$a = x5;$$

$$b = x6;$$

$$c = x7;$$

$$h = x8;$$

addi x8, x7, -5

add x9, x5, x6

Sub x8, x9, x8

(2.3). A given application written in Java runs 40 seconds on a desktop processor. A new Java compiler is released that inquires only 0.7 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1 time.

How long can we expect the application to run using the new compiler?

$$\text{time}_o = IC_o * CPI_o * T_{t_o}$$

$$\text{time}_n = 0.7 IC_o * 1.1 CPI_o * T_{t_n} = 0.7 \times 1.1 \times 40 = 30.8 \text{ seconds}$$

(2.4). Translate the c statement  $A[10] = h + A[4]$  to RISC-V assembly instructions, assuming  $h$  in  $x10$ , base address of  $A$  in  $x18$ , and each array element is a word (32 bit) (RV-32 is assumed here).
$$h = x10;$$
  

$$\text{base } A = x18;$$

LW x9, 16(x18)

Add x9, x9, x10

SW x9, 80(x18)

- 6 (2.5). Suppose a program runs in 100 seconds on a computer, with **multiply operations** account for 70 seconds of the time. How much improvement in **multiply operations'** performance to get the program to run 2 times faster?

$$\frac{70}{n} + 30 = \frac{100}{2} \rightarrow \frac{70/n + 30}{-30} = \frac{50}{-30}$$

$$70/n = 20$$

$$n = 3.5$$

- 6 (2.6). When determine the CPU performance, CPU time is related to Instruction count, CPI and clock rate. Please fill out the following table to check which aspects are affected by the specific hardware and software component.

|                      | Instruction Count | CPI | Clock Rate |
|----------------------|-------------------|-----|------------|
| Algorithm            | ✓                 | ✓   |            |
| ISA                  | ✓                 | ✓   | ✓          |
| Compiler             | ✓                 | ✓   |            |
| Programming language | ✓                 | ✓   |            |

- 6 (2.7). Given a 32-bit binary number, please demonstrate **the range** it could represent in 2 different cases.

(a) case 1: unsigned integer  $[0, 2^{32}-1]$

(b) case 2: signed integer  $[-2^{31}, 2^{31}-1]$

$$IC \begin{cases} \text{class A} = (1 \times 10^6)(0.40) = 4 \times 10^5 \\ \text{class B} = (1 \times 10^6)(0.30) = 3 \times 10^5 \\ \text{class C} = (1 \times 10^6)(0.20) = 2 \times 10^5 \\ \text{class D} = (1 \times 10^6)(0.10) = 10^5 \end{cases}$$

$$P1 \begin{cases} CR = 2.56 \text{ GHz} \\ CPI = 1, 2, 3, 4 \end{cases}$$

$$P2 \begin{cases} CR = 3 \text{ GHz} \\ CPI = 2, 2, 2, 2 \end{cases}$$

(12pts) 3. Consider two different implementation of the same instruction set architecture. The instruction can be divided into four classes according to their CPI (class A, B, C, and D), Implementation P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 4. Implementation P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of  $1.0E6$  instructions divided into classes as follows: 40% class A; 30% class B; 20% class C; 10% class D

- (a) What is the global CPI for each implementation?  
(b) Which implementation is faster: P1 or P2? By how much?

$$P1 \text{ Global CPI} = (4 \times 10^5)(1) + (3 \times 10^5)(2) + (2 \times 10^5)(3) + (10^5)(4) = 2 \times 10^6 = \text{Global CPI } P1$$

$$P2 \text{ Global CPI} = (4 \times 10^5)(2) + (3 \times 10^5)(2) + (2 \times 10^5)(2) + (10^5)(2) = 2 \times 10^6 = \text{Global CPI } P2$$

$$\text{Time } P1 = \frac{2 \times 10^6}{(2.5 \times 10^9)} = 8 \times 10^{-4}$$

$$\frac{8 \times 10^{-4}}{6.66 \times 10^{-4}} = 1.2$$

$$\text{Time } P2 = \frac{2 \times 10^6}{(3 \times 10^9)} = 6.66 \times 10^{-4}$$

$P2$  is 1.2 times faster

(10 pts) 4. Please provide the instruction type (2pts), binary (6pts), and hexadecimal (2pts) representation of the following instruction, the table below is given for your reference.

| Instruction          | Format | funct7     | rs2 | rs1 | funct3 | rd         | opcode  |
|----------------------|--------|------------|-----|-----|--------|------------|---------|
| add (add)            | R      | 0000000    | reg | reg | 000    | reg        | 0110011 |
| sub (sub)            | R      | 0100000    | reg | reg | 000    | reg        | 0110011 |
| Instruction          | Format | immediate  |     | rs1 | funct3 | rd         | opcode  |
| addi (add immediate) | I      | constant   |     | reg | 000    | reg        | 0010011 |
| lw (load word)       | I      | address    |     | reg | 010    | reg        | 0000011 |
| Instruction          | Format | immed-iate | rs2 | rs1 | funct3 | immed-iate | opcode  |
| sw (store word)      | S      | address    | reg | reg | 010    | address    | 0100011 |

Lw x3, 100 (x22)

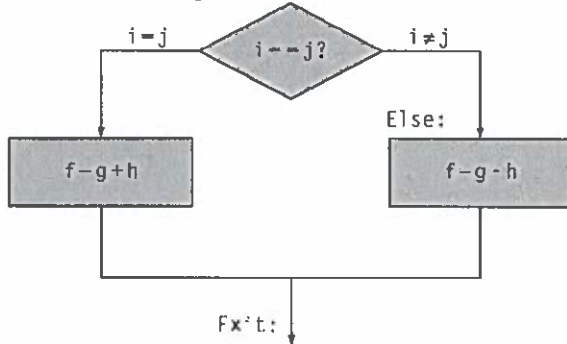
• I-type

• 0010 0101

10 (10 pts) 5. Please change the following c statement to RISC-V statements

if (i==j) {f = g+h} else {f = g-h};

Assume the register allocations are: i: x20; j: x21; f: x22; g: x23; h: x24



bne x20, x21, Else  
add x22, x23, x24  
beq x0, x0, Exit  
Else: sub x22, x23, x24  
Exit:

i = x20;  
j = x21;  
f = x22;  
g = x23;  
h = x24;

10 (12 pts) 6. Please change the following c statement to RISC-V statements.

Assume that base address of array A is in register x10, base address of array B is in register x11, (assuming each array element is a doubleword (64 bit), each register is also 64-bit wide). Variables register allocation are: c: x6, i: x7, and j: x8.

What is the RISC-V assembly code for the C statement?

c = A[i] - B[j];

Slli x7, x7, 3  
add x10, x10, x7  
ld x7, 0(x10)  
Slli x8, x8, 3  
add x11, x11, x8  
ld x8, 0(x11)  
sub x6, x7, x8

Register Allocation

0.5 (8 pts) 7. Suppose the program counter (PC) is set to 0x4000 0000

(4 pts)(a) What range of addresses can be reached using RISC-V jump-and-link (jal) instruction?

$[0, 2^8 - 1]$

(4 pts)(b) What range of addresses can be reached using RISC-V branch if equal (beq) instruction?

$[-2^7, 2^7 - 1]$



### RISC-V assembly language

| Category      | Instruction                      | Example           | Meaning                            | Comments                                   |
|---------------|----------------------------------|-------------------|------------------------------------|--------------------------------------------|
| Arithmetic    | Add                              | add x5, x6, x7    | $x5 = x6 + x7$                     | Three register operands                    |
|               | Subtract                         | sub x5, x6, x7    | $x5 = x6 - x7$                     | Three register operands                    |
|               | Add immediate                    | addi x5, x6, 20   | $x5 = x6 + 20$                     | Used to add constants                      |
| Data transfer | Load doubleword                  | ld x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Doubleword from memory to register         |
|               | Store doubleword                 | sd x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Doubleword from register to memory         |
|               | Load word                        | lw x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Word from memory to register               |
|               | Load word, unsigned              | lwu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Unsigned word from memory to register      |
|               | Store word                       | sw x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Word from register to memory               |
|               | Load halfword                    | lh x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Halfword from memory to register           |
|               | Load halfword, unsigned          | lhu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Unsigned halfword from memory to register  |
|               | Store halfword                   | sh x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Halfword from register to memory           |
|               | Load byte                        | lb x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Byte from memory to register               |
|               | Load byte, unsigned              | lbu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Byte halfword from memory to register      |
|               | Store byte                       | sb x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Byte from register to memory               |
|               | Load reserved                    | lr.d x5, (x6)     | $x5 = \text{Memory}[x6]$           | Load; 1st half of atomic swap              |
|               | Store conditional                | sc.d x7, x5, (x6) | $\text{Memory}[x6] = x5; x7 = 0/1$ | Store; 2nd half of atomic swap             |
|               | Load upper immediate             | lui x5, 0x12345   | $x5 = 0x12345000$                  | Loads 20-bit constant shifted left 12 bits |
| Logical       | And                              | and x5, x6, x7    | $x5 = x6 \& x7$                    | Three reg. operands; bit-by-bit AND        |
|               | Inclusive or                     | or x5, x6, x8     | $x5 = x6   x8$                     | Three reg. operands; bit-by-bit OR         |
|               | Exclusive or                     | xor x5, x6, x9    | $x5 = x6 \wedge x9$                | Three reg. operands; bit-by-bit XOR        |
|               | And immediate                    | andi x5, x6, 20   | $x5 = x6 \& 20$                    | Bit-by-bit AND reg. with constant          |
|               | Inclusive or immediate           | ori x5, x6, 20    | $x5 = x6   20$                     | Bit-by-bit OR reg. with constant           |
|               | Exclusive or immediate           | xori x5, x6, 20   | $x5 = x6 \wedge 20$                | Bit-by-bit XOR reg. with constant          |
| Shift         | Shift left logical               | sll x5, x6, x7    | $x5 = x6 \ll x7$                   | Shift left by register                     |
|               | Shift right logical              | srl x5, x6, x7    | $x5 = x6 \gg x7$                   | Shift right by register                    |
|               | Shift right arithmetic           | sra x5, x6, x7    | $x5 = x6 \ggg x7$                  | Arithmetic shift right by register         |
|               | Shift left logical immediate     | slli x5, x6, 3    | $x5 = x6 \ll 3$                    | Shift left by immediate                    |
|               | Shift right logical immediate    | srlr x5, x6, 3    | $x5 = x6 \gg 3$                    | Shift right by immediate                   |
|               | Shift right arithmetic immediate | srair x5, x6, 3   | $x5 = x6 \ggg 3$                   | Arithmetic shift right by immediate        |

|                      |                                   |                  |                            |                                                  |
|----------------------|-----------------------------------|------------------|----------------------------|--------------------------------------------------|
| Conditional branch   | Branch if equal                   | beq x5, x6, 100  | if (x5 == x6) go to PC+100 | PC-relative branch if registers equal            |
|                      | Branch if not equal               | bne x5, x6, 100  | if (x5 != x6) go to PC+100 | PC-relative branch if registers not equal        |
|                      | Branch if less than               | blt x5, x6, 100  | if (x5 < x6) go to PC+100  | PC-relative branch if registers less             |
|                      | Branch if greater or equal        | bge x5, x6, 100  | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
|                      | Branch if less, unsigned          | bltu x5, x6, 100 | if (x5 < x6) go to PC+100  | PC-relative branch if registers less             |
|                      | Branch if greater/equal, unsigned | bgeu x5, x6, 100 | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
| Unconditional branch | Jump and link                     | jal x1, 100      | x1 = PC+4; go to PC+100    | PC-relative procedure call                       |
|                      | Jump and link register            | jalr x1, 100(x5) | x1 = PC+4; go to x5+100    | Procedure return; indirect call                  |

| Name<br>(Field Size) | 7 bits                      | 5 bits | 5 bits | 3 bits | 5 bits        | 7 bits | Comments                      |
|----------------------|-----------------------------|--------|--------|--------|---------------|--------|-------------------------------|
| R-type               | funct7                      | rs2    | rs1    | funct3 | rd            | opcode | Arithmetic instruction format |
| I-type               | immediate[11:0]             |        | rs1    | funct3 | rd            | opcode | Loads & immediate arithmetic  |
| S-type               | immed[11:5]                 | rs2    | rs1    | funct3 | immed[4:0]    | opcode | Stores                        |
| SB-type              | immed[12,10:5]              | rs2    | rs1    | funct3 | immed[4:1,11] | opcode | Conditional branch format     |
| UJ-type              | immediate[20,10:1,11,19:12] |        |        |        | rd            | opcode | Unconditional jump format     |
| U-type               | immediate[31:12]            |        |        |        | rd            | opcode | Upper immediate format        |

Information page

**BELOW AVERAGE**



**Below Average Student Original Graded Assignment**

**ELEG 4339: Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Exam 1

Spring 2022

Name: (Print) Ryan Johnson

(Sign) 

|              |            |           |
|--------------|------------|-----------|
| 1            | 16         | 11.5      |
| 2            | 42         | 28.5      |
| 3            | 12         | 9         |
| 4            | 10         | 5         |
| 5            | 10         | 10        |
| 6            | 12         | 0         |
| 7            | 8          | 0         |
|              |            |           |
| <b>Total</b> | <b>110</b> | <b>64</b> |

Instructions:

You have 80 minutes to complete this exam. This exam counts for 15% of your course grade.

11.5  
1.5

(16pts) 1. Short answer (2 points each)

(1.1). Computer architect divide processors based on design criteria of instruction sets into two broad categories: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Choose **all** the following characteristics that **are applicable to RISC processor**.

- a. Variable-length instructions
- ☒ b. Fixed-length instructions
- c. Many instructions (typically, hundreds)
- ☒ d. Strives for a minimum instruction set
- ☒ e. Each instruction performs a basic computation
- f. Each instruction can perform an arbitrarily complex computation

2 (1.2). Do the following changes to a computer system increase throughput, decrease response time, or both?

Replacing the processor in a computer with a faster version.

both

0 (1.3) Please describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

2 (1.4). Representing  $-5_{10}$  in 8-bit binary number using 2's complement

$$\begin{array}{r} 5 = 0000\ 0101 \\ 1111\ 1010 \\ \hline 1111\ 1011 \end{array}$$

$$-5 = 1111\ 1011$$

2 (1.5) Representing  $-5_{10}$  in 32-bit binary number using 2's complement

$$\begin{array}{r} 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101 \\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010 \\ \hline -5 = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011 \end{array}$$

2 (1.6) Which are five classic components of a computer?

input, output, memory, datapath, control

2 (1.7) For a given function, which programming language likely takes more lines of code?

- (a) C      (b) RISC-V Assembly language

0 (1.8) What is ISA? Please give one example here.

(42 pts) 2. Medium answer (6 points each)

(2.1). A program requires executing 100 million instructions on a processor which typically with CPI of 2 and a 2GHz clock. How much time will the program take?

3.5

$$\frac{10^8 \times 2}{2 \times 10^9} = 1 \text{ sec}$$

(2.2). Assuming variable  $a$  is stored in register  $x5$ ,  $b$  is stored in  $x6$ ,  $c$  is stored in  $x7$ , and the result  $h$  is to be stored in  $x8$ , what is the RISC-V assembly instructions equivalent to the C statement

$h = (a+b) - (c-5)$

addi x20, x7, -5

add x21, x5, x6

sub x8, x21, x20

order -

(2.3). A given application written in Java runs 40 seconds on a desktop processor. A new Java compiler is released that requires only 0.7 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1 times.

How long can we expect the application to run using the new compiler?

0.7 x 1.1

(2.4). Translate the c statement  $A[10] = h + A[4]$  to RISC-V assembly instructions, assuming  $h$  in  $x10$ , base address of  $A$  in  $x18$ , and each array element is a word (32 bit) (RV-32 is assumed here).

lw x9, 4(x18)  
add x9, x10, x9  
sw x9, 4(x18)

b (2.5). Suppose a program runs in 100 seconds on a computer, with **multiply operations** account for 70 seconds of the time. How much improvement in **multiply operations'** performance to get the program to run 2 times faster?

$$100 = \frac{70}{n} + 30$$

$$\frac{10}{2} = \frac{70}{n+30}$$

$$50 = \frac{70}{n} + 30$$

$$20n = 70$$

$$n = 3.5$$

b (2.6). When determine the CPU performance, CPU time is related to Instruction count, CPI and clock rate. Please fill out the following table to check which aspects are affected by the specific hardware and software component.

|                      | Instruction Count | CPI | Clock Rate |
|----------------------|-------------------|-----|------------|
| Algorithm            | ✓                 | ✓   |            |
| ISA                  | ✓                 | ✓   | ✓          |
| Compiler             | ✓                 | ✓   |            |
| Programming language | ✓                 | ✓   |            |

5.5  
3 (2.7). Given a 32-bit binary number, please demonstrate the range it could represent in 2 different cases.

(a) case 1: unsigned integer

$$2^{31} = 2147483648$$

$$\text{Range } (0, 2147483648)$$

(b) case 2: signed integer

$$[0, 2^{32}-1]$$

$$2^{-31} = -2147483648$$

$$\text{Range } (-2147483648, 2147483647)$$

exponent will work

9 (12pts) 3. Consider two different implementation of the same instruction set architecture. The instruction can be divided into four classes according to their CPI (class A, B, C, and D), Implementation P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 4. Implementation P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 40% class A; 30% class B; 20% class C; 10% class D

- (a) What is the global CPI for each implementation?  
(b) Which implementation is faster: P1 or P2? By how much?

$$CPI P1 = \frac{(1 \times .4) + (2 \times .3) + (3 \times .2) + (4 \times .1)}{2.5 \times 10^9} \times 10^6 = 0.8 \times 10^{-3}$$

$$CPI P2 = \frac{(2 \times .4) + (2 \times .3) + (2 \times .2) + (2 \times .1)}{3 \times 10^9} \times 10^6 = 0.667 \times 10^{-3}$$

$$Global CPI P1 = \frac{(0.8 \times 10^{-3}) \times (2.5 \times 10^9)}{1 \times 10^6} = 2$$

$$Global CPI P2 = \frac{(0.667 \times 10^{-3}) \times (3 \times 10^9)}{1 \times 10^6} = 2.001$$

P1 is faster by 0.001 seconds

5 (10 pts) 4. Please provide the instruction type (2pts), binary (6pts), and hexadecimal (2pts) representation of the following instruction, the table below is given for your reference.

| Instruction          | Format | funct7     | rs2 | rs1 | funct3 | rd         | opcode  |
|----------------------|--------|------------|-----|-----|--------|------------|---------|
| add (add)            | R      | 0000000    | reg | reg | 000    | reg        | 0110011 |
| sub (sub)            | R      | 0100000    | reg | reg | 000    | reg        | 0110011 |
| Instruction          | Format | immediate  |     | rs1 | funct3 | rd         | opcode  |
| addi (add immediate) | I      | constant   |     | reg | 000    | reg        | 0010011 |
| lw (load word)       | I      | address    |     | reg | 010    | reg        | 0000011 |
| Instruction          | Format | immed-iate | rs2 | rs1 | funct3 | immed-iate | opcode  |
| sw (store word)      | S      | address    | reg | reg | 010    | address    | 0100011 |

Lw x3, 100 (x22)

I type

opcode = 0000011

funct3 = 010

immediate = 01100010

$$rs1 = 16 + 4 + 2 = 10110$$

rd = 0

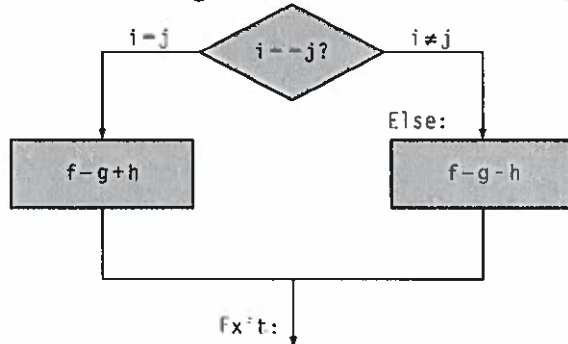
0000 0011 1001 1001 0110 0000 1011 0000 0110

C92B038

10 (10 pts) 5. Please change the following c statement to RISC-V statements

if (i==j) {f = g+h} else {f = g-h};

Assume the register allocations are: i: x20; j: x21; f: x22; g: x23; h: x24



bne x20/x21,Else  
 add x22,x23,x24  
 beq x0/x0,Exit  
 Else: sub x22,x23,x24  
 Exit!

0 (12 pts) 6. Please change the following c statement to RISC-V statements.

Assume that base address of array A is in register x10, base address of array B is in register x11, (assuming each **array element is a doubleword (64 bit)**, each register is also **64-bit wide**). Variables register allocation are: c: x6, i: x7, and j: x8.

What is the RISC-V assembly code for the C statement?

c = A[i] - B[j]

add x6,

(8 pts) 7. Suppose the program counter (PC) is set to 0x 4000 0000

(4 pts)(a) What range of addresses can be reached using RISC-V jump-and-link (jal) instruction?

(4 pts)(b) What range of addresses can be reached using RISC-V branch if equal (beq) instruction?



### RISC-V assembly language

| Category      | Instruction                      | Example           | Meaning                            | Comments                                   |
|---------------|----------------------------------|-------------------|------------------------------------|--------------------------------------------|
| Arithmetic    | Add                              | add x5, x6, x7    | $x5 = x6 + x7$                     | Three register operands                    |
|               | Subtract                         | sub x5, x6, x7    | $x5 = x6 - x7$                     | Three register operands                    |
| Data transfer | Add immediate                    | addi x5, x6, 20   | $x5 = x6 + 20$                     | Used to add constants                      |
|               | Load doubleword                  | ld x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Doubleword from memory to register         |
|               | Store doubleword                 | sd x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Doubleword from register to memory         |
|               | Load word                        | lw x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Word from memory to register               |
|               | Load word, unsigned              | lwu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Unsigned word from memory to register      |
|               | Store word                       | sw x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Word from register to memory               |
|               | Load halfword                    | lh x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Halfword from memory to register           |
|               | Load halfword, unsigned          | lhu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Unsigned halfword from memory to register  |
|               | Store halfword                   | sh x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Halfword from register to memory           |
|               | Load byte                        | lb x5, 40(x6)     | $x5 = \text{Memory}[x6 + 40]$      | Byte from memory to register               |
|               | Load byte, unsigned              | lbu x5, 40(x6)    | $x5 = \text{Memory}[x6 + 40]$      | Byte halfword from memory to register      |
|               | Store byte                       | sb x5, 40(x6)     | $\text{Memory}[x6 + 40] = x5$      | Byte from register to memory               |
|               | Load reserved                    | lr.d x5, (x6)     | $x5 = \text{Memory}[x6]$           | Load: 1st half of atomic swap              |
|               | Store conditional                | sc.d x7, x5, (x6) | $\text{Memory}[x6] = x5; x7 = 0/1$ | Store: 2nd half of atomic swap             |
|               | Load upper immediate             | lui x5, 0x12345   | $x5 = 0x12345000$                  | Loads 20-bit constant shifted left 12 bits |
| Logical       | And                              | and x5, x6, x7    | $x5 = x6 \& x7$                    | Three reg. operands; bit-by-bit AND        |
|               | Inclusive or                     | or x5, x6, x8     | $x5 = x6   x8$                     | Three reg. operands; bit-by-bit OR         |
|               | Exclusive or                     | xor x5, x6, x9    | $x5 = x6 \wedge x9$                | Three reg. operands; bit-by-bit XOR        |
|               | And immediate                    | andi x5, x6, 20   | $x5 = x6 \& 20$                    | Bit-by-bit AND reg. with constant          |
|               | Inclusive or immediate           | ori x5, x6, 20    | $x5 = x6   20$                     | Bit-by-bit OR reg. with constant           |
|               | Exclusive or immediate           | xori x5, x6, 20   | $x5 = x6 \wedge 20$                | Bit-by-bit XOR reg. with constant          |
| Shift         | Shift left logical               | sll x5, x6, x7    | $x5 = x6 \ll x7$                   | Shift left by register                     |
|               | Shift right logical              | srl x5, x6, x7    | $x5 = x6 \gg x7$                   | Shift right by register                    |
|               | Shift right arithmetic           | sra x5, x6, x7    | $x5 = x6 \gg x7$                   | Arithmetic shift right by register         |
|               | Shift left logical immediate     | slli x5, x6, 3    | $x5 = x6 \ll 3$                    | Shift left by immediate                    |
|               | Shift right logical immediate    | srlr x5, x6, 3    | $x5 = x6 \gg 3$                    | Shift right by immediate                   |
|               | Shift right arithmetic immediate | srair x5, x6, 3   | $x5 = x6 \gg 3$                    | Arithmetic shift right by immediate        |

|                      |                                   |                  |                            |                                                  |
|----------------------|-----------------------------------|------------------|----------------------------|--------------------------------------------------|
| Conditional branch   | Branch if equal                   | beq x5, x6, 100  | if (x5 == x6) go to PC+100 | PC-relative branch if registers equal            |
|                      | Branch if not equal               | bne x5, x6, 100  | if (x5 != x6) go to PC+100 | PC-relative branch if registers not equal        |
|                      | Branch if less than               | blt x5, x6, 100  | if (x5 < x6) go to PC+100  | PC-relative branch if registers less             |
|                      | Branch if greater or equal        | bge x5, x6, 100  | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
|                      | Branch if less, unsigned          | bltu x5, x6, 100 | if (x5 < x6) go to PC+100  | PC-relative branch if registers less             |
|                      | Branch if greater/equal, unsigned | bgeu x5, x6, 100 | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal |
| Unconditional branch | Jump and link                     | jal x1, 100      | x1 = PC+4; go to PC+100    | PC-relative procedure call                       |
|                      | Jump and link register            | jalr x1, 100(x5) | x1 = PC+4; go to x5+100    | Procedure return; indirect call                  |

| Name<br>(Field Size) | Field                       | Field  | Field  | Field  | Field         | Field  | Comments                      |
|----------------------|-----------------------------|--------|--------|--------|---------------|--------|-------------------------------|
|                      | 7 bits                      | 5 bits | 5 bits | 3 bits | 5 bits        | 7 bits |                               |
| R-type               | funct7                      | rs2    | rs1    | funct3 | rd            | opcode | Arithmetic instruction format |
| I-type               | immediate[11:0]             |        | rs1    | funct3 | rd            | opcode | Loads & immediate arithmetic  |
| S-type               | immed[11:5]                 | rs2    | rs1    | funct3 | immed[4:0]    | opcode | Stores                        |
| SB-type              | immed[12,10:5]              | rs2    | rs1    | funct3 | immed[4:1,11] | opcode | Conditional branch format     |
| UJ-type              | immediate[20,10:1,11,19:12] |        |        |        | rd            | opcode | Unconditional jump format     |
| U-type               | immediate[31:12]            |        |        |        | rd            | opcode | Upper immediate format        |

Information page



# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**SPRING 2022**

**Title of Assignment: EXAM2**

## **TOPICS COVERED IN EXAM2**

**Exam 2 covered the following topics:**

1. Signed and unsigned number,
2. Arithmetic for computer: Addition, subtraction, multiplication
3. Floating point representation, IEEE 754 standard
4. Overflow and underflow
5. The Processor: datapath and control
6. A simple single cycle implementation scheme; multi-cycle implementation
7. Enhancing performance with Pipelining; data, structural, and control hazards

# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

## **Exam 2**

**ABOVE AVERAGE**

**Above Average Student Original Graded Assignment**

**ELEG 4339: Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Exam 2

Spring 2022

Name: (Print) Xavier Dukes  
(Sign) Xavier Dukes

|              |     |      |
|--------------|-----|------|
| 1            | 17  | 16.5 |
| 2            | 30  | 20   |
| 3            | 10  | 8    |
| 4            | 10  | 5    |
| 5            | 19  | 17   |
| 6            | 14  | 14   |
|              |     |      |
| <b>Total</b> | 100 | 90.5 |

Instructions:

You have 80 minutes to complete the exam. The exam totals 100 points; the points for each problem are indicated. This exam counts for 15% of your course grade.

(17pts) 1. Short answer

(2pt)(1.1). Which of the following belongs to **normalized scientific notation**.

- a.  $27.35 \times 10^9$       b.  $2.735 \times 10^9$       c.  $0.2735 \times 10^9$

(2pt) (1.2). Is the IEEE 754 single-precision floating point number

1011 0001 1000 1110 1111 0000 0000 0000

- a. positive      b. negative

(2pt) (1.3). Name the **special purpose register** that stores the **address** of the next instruction to be fetched from memory.

Answer: Program Counter (PC)

(2pt) (1.4). In the integer multiplication, if we ignore the sign bits, what is the length of the product of a **n-bit** multiplicand and an **m-bit** multiplier?

- a. n    b. m    c.  $2n$     d.  $2m$     e.  $n+m$     f.  $n-m$     g. None of the above

(3pt) (1.5) What is overflow and underflow for floating point arithmetic?

overflow → values that exceeded greatest possible magnitude that is representable

underflow → values too small / close to zero that they can't be represented

(3pt) (1.6) Of all the available datapath components, please specify (place a checkmark) whether the component will be needed for the execution of instruction: `ld x1, 400(x2)`

- | Yes                                 | NO                       |                           |
|-------------------------------------|--------------------------|---------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | Instruction memory        |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | ALU                       |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | Register file             |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | Immediate generation unit |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | Data memory               |

(3pt) (1.7). For the following RISC-V instructions, they will use the arithmetic-logic unit (ALU) after reading the registers, please explain what is the function of ALU for each instruction.

|                                    | ALU function                                                                                         |
|------------------------------------|------------------------------------------------------------------------------------------------------|
| a: <code>ADD x15, x16, x19</code>  | Perform Addition $x16 + x19$ The result is in $x15$                                                  |
| b: <code>SD x2, 200(x1)</code>     | Calculate mem. Address value in $x1$                                                                 |
| c: <code>BEQ x3, x4, offset</code> | Decision to take branch<br>Sees if $x3 \neq x4$ Values are equal<br>Subtracts if results then branch |

(30pts) 2.

(10 pts) (2.1) Given a 32-bit binary number, please demonstrate the range it could represent in 3 different cases.

(2 pts)(a) case 1: unsigned integer

$$0, 2^n - 1 \rightarrow [0, 2^{32} - 1]$$

(2 pts) (b) case 2: signed integer

$$[-2^{n-1}, 2^{n-1} - 1] \rightarrow [-2^{31}, 2^{31} - 1]$$

(6 pts) (c) case 3: single-precision IEEE 754 floating-point representation (please show the reasoning).

Exponent Value

0000 ...

1111 ...

Don't use

Special Case

1 + all decimals = almost 2

Largest Magnitude

$$\pm 1.111111...1 * 2^{254-127}$$

$$\approx \pm 2.0 * 2^{127}$$

Smallest Magnitude

$$\pm 1.000...0 * 2^{1-127}$$

$$\pm 1.000...0 * 2^{-126}$$

(10 pts) (2.2) What is the decimal value of this IEEE 754 single-precision floating point number

1 8 23  
1011 0001 1000 1110 1111 0000 0000 0000

Exponent 01100011 = 99

$$-(1 + .0001110111) * 2^{99-127}$$

$$-1.0001110111 * 2^{-28}$$

(10pts) (2.3) Please represent the decimal number: -213.75 using single-precision floating point IEEE 754 standard.

$$x = (-1)^s * (1 + \text{Fraction}) * 2^{(\text{Exponent} - \text{Bias})}$$

$$(-1)^1 * (1 + .10101011) * 2^{134-127}$$

1 10000110 10101011 0000000000000000

128 64 32 16 8 4 2  
1 1 0 1 0 1 0

$$0.75 * 2 = 1.5$$

$$.5 * 2 = 1.0$$

1.10101011

$$1.10101011 * 2^{7+127}$$

drop first one b/c implied

### (10 pts) 3. Overflow

Assume the registers x5 and x6 hold the values as follows:

x5: 0x B000 0000 0000 0000

x6: 0x 9000 0000 0000 0000

Carry out B  
1 + 9  
↳ 15 000...000

(3.1) (i) what is the value of x30 after execution of the following assembly code?

(ii) Is the result in x30 the desired result, or has there been overflow?

Add x30, x5, x6

Produces overflow

i) 0x5000 0000 0000 0000

ii) No this is not the desired result as it produces an unaccounted carry bit 1

(3.2) (i) what is the value of x30 after execution of the following assembly code?

(ii) Is the result in x30 the desired result, or has there been overflow?

Sub x30, x5, x6

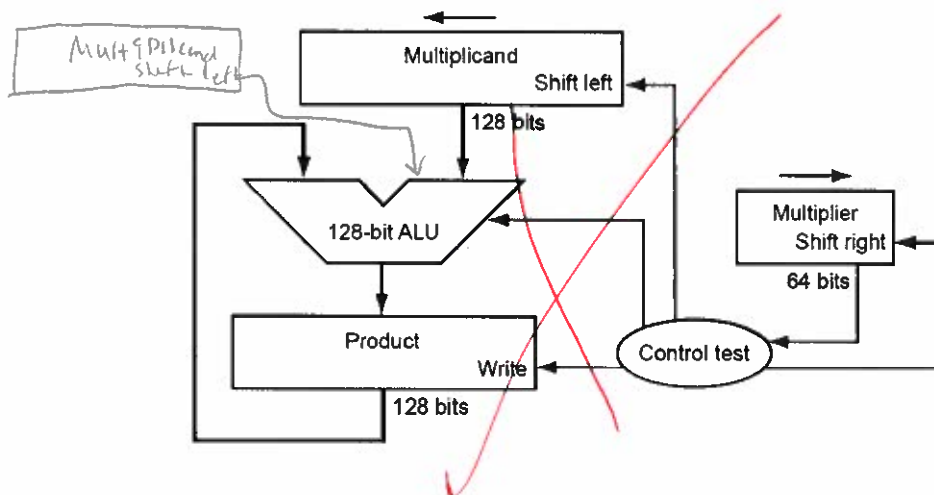
0x130  
- 0x90  
0x20

i) 0x2000 0000 0000 0000

ii) yes this is correct. No overflow can occur with values that are same sign during subtraction op

(10pts) 4. Integer multiplication. We will use 8-bit binary number as a case study

(5pts) (4.1) the following diagram demonstrate the first version of multiplication of two 64-bit binary number, Please revise it to perform the two 8-bit binary number multiplication.



(5pts) (4.2) Please briefly explain how many clock cycles do you need to perform a 8-bit multiplication following above diagram. Please explain the reasoning.

it takes  $3 * n$  to perform multiplication operations

$3 * 8 = 24$  clock cycles to complete the multiplication

- n is number of bits



17

**(19 pts) 5. Single cycle implementation – multi cycle implementation – pipelined implementation:**

Calculate cycle time assuming negligible delays (for muxes, control unit, immediate Gen, PC access, shift left 1, wires) except: Instruction and Data Memory (200ps); ALU and adders (200ps); and Register File access (reads or writes) (100 ps)

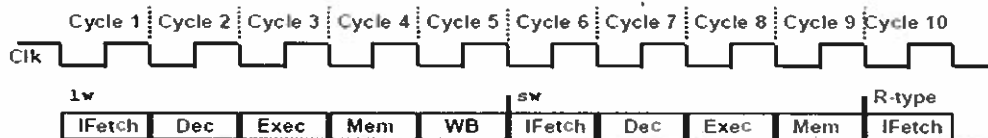
4

- (5 pts) (5.1) (i) Please fill in the table and calculate the total time needed for each instruction  
(ii) What will be a good time for a single cycle time?

| Instruction Class    | Instruction memory | Register Read | ALU Operation | Data Mem | Register Write | Total  |
|----------------------|--------------------|---------------|---------------|----------|----------------|--------|
| R type( such as SUB) | 200                | 100           | 200           | X        | 100            | 600 Ps |
| Load (LW)            | 200                | 100           | 200           | 200      | 100            | 800 Ps |
| Store (SW)           | 200                | 100           | 200           | 200      | X              | 700 Ps |
| BNE                  | 200                | 100           | 200           | X        | X              | 500 Ps |

2

- (2 pts) (5.2) when we adopt a multicycle implementation. Each step will take 1 clock cycle, please explain what will be an appropriate cycle time?



200ps Per Stage as this is the longest stage time

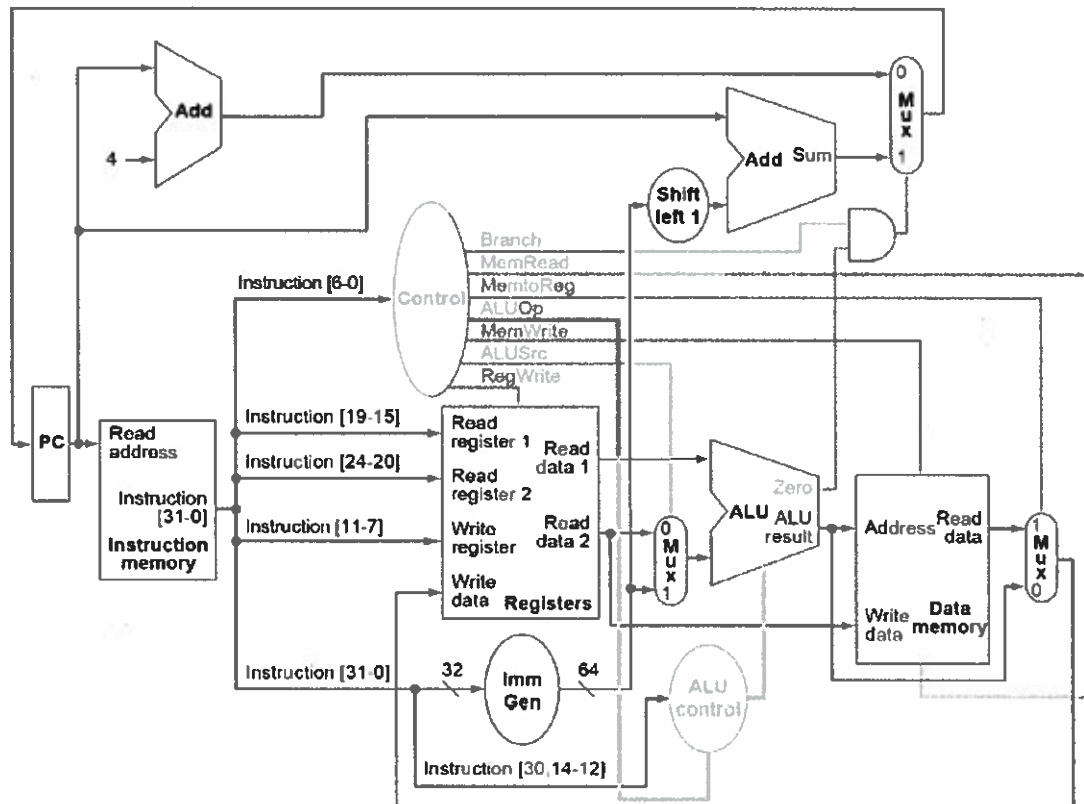
4

**(4 pts) (5.3)**

- (i) Please explain what is the pipelined implementation.
- (ii) If a new technology could shorten ALU operation by 20%, will it affect the speedup obtained from pipelining? Please briefly explain. No because a clock cycle depends on longest delay & it will still be 200ps for Data M
- (iii) What if the ALU operation now take 20% more time? Please briefly explain. This will negatively affect speedup as now each stage must be 240ps long for new longest delay

i) balanced stages

- one instruction is being executed while another is being fetched this increases throughput



(4 pts) 5.4. Please fill in the following table, which specify how to set the control signals when executing the 2 instructions.

| Instruction | Branch                              | MemtoReg                            | ALUSrc                              | RegWrite                            | MemRead                             | MemWrite                            | ALUOp |
|-------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------|
| SD          | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | 00    |
| ADDI        | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | 01    |

(4 pts) 5.5. Please explain the detailed execution steps for the instruction:

ld x10, 100(x11)

the Instruction is fetched ; decoded then during execution the memory address is calculated with value in X11 ; immediate value that is generated to proper length. Accesses calculated address reads the memory there, performs mem to Reg the RegWrite part as the mem. value is stored in register X10

(14 pts) 6. The following RISC-V code segment is corresponding to the C code:  
 $A = B + E$ ;  $C = B - F$ ;

1. ld x10, 0(x6)  
 2. ld x11, 8(x6)  
 3. add x12, x10, x11  
 4. sd x12, 16(x6)  
 5. ld x13, 24(x6)  
 6. sub x15, x10, x13  
 7. sd x15, 40(x6)

$$4 + 7 = 11$$

Cycles if Perfect

Need 2 waits on →  
 3 3/6

extra  
 $4 + 7 + 2 = 13$

(2 pts) 6.1 Please explain what is data hazard. Please give an example from the code sequence above.  
 Data hazard - when the value of a previous operation isn't ready to be used by next instruction.

EX. Step 2 → 3 is an example as the value of x11 isn't updated until WB stage, so if pipelined the add must wait

(4 pts) 6.2 Assume forwarding (aka bypassing) scheme is adopted here. (i) Please specify the data hazards even with the forwarding scheme and point out where stalls should be added. (ii) how many clock cycles do you need to implement the above code sequence assuming a 5-stage pipeline?

i) Steps 2 → 3, 5 → 6, 3 → 6 must wait on even with bypassing because not available until WB

ii) you need 13 clock cycles for the code above

(4 pts) 6.3 (i) Please reorder the code segment (while do not change the function of the program) to avoid the stalls. (ii) how many clock cycles do you need to implement the above code sequence?

assuming  
 bypassing  
 hardware  
 i) ld x11, 8(x6)  
 ld x10, 0(x6)  
 ld x13, 24(x6)  
 add x12, x10, x11  
 sd x12, 16(x6)  
 sub x15, x10, x13  
 sd x15, 40(x6)

ii) Need 11 clock cycles

(4 pts) 6.4 What is the minimum number of cycles needed to completely execute N instructions on a CPU with a M stage pipeline? Justify your formula.

$$(M_{\text{stages}} - 1) + N_{\text{number Instructions}} = \text{Clock Cycles}$$

because after the first  $(M_{\text{stages}} - 1)$  1 instruction will be completed every clock cycle, assuming Perfect

**AVERAGE**

## **Average Student Original Graded Assignment**

**ELEG 4339: Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Exam 2

Spring 2022

Name: (Print) Jacqueline Torralba

(Sign) 

|              |     |      |
|--------------|-----|------|
| 1            | 17  | 17   |
| 2            | 30  | 25   |
| 3            | 10  | 3.5  |
| 4            | 10  | 4.5  |
| 5            | 19  | 17   |
| 6            | 14  | 12.5 |
|              |     |      |
| <b>Total</b> | 100 | 79.5 |

Instructions:

You have 80 minutes to complete the exam. The exam totals 100 points; the points for each problem are indicated. This exam counts for 15% of your course grade.

17

(17pts) 1. Short answer

(2pt)(1.1). Which of the following belongs to **normalized scientific notation**.

a.  $27.35 \times 10^9$

b.  $2.735 \times 10^9$

c.  $0.2735 \times 10^9$

(2pt) (1.2). Is the IEEE 754 single-precision floating point number

1011 0001 1000 1110 1111 0000 0000 0000

a. positive

b. negative

(2pt) (1.3). Name the **special purpose register** that stores the **address** of the next instruction to be fetched from memory.

Answer: Adder, PC

(2pt) (1.4). In the integer multiplication, if we ignore the sign bits, what is the length of the product of a **n-bit** multiplicand and an **m-bit** multiplier?

a. n b. m c.  $2n$  d.  $2m$  e.  $n+m$  f.  $n-m$  g. None of the above

(3pt) (1.5) What is overflow and underflow for floating point arithmetic?

- overflow would be when the positive exponent is too large to fit in the exponent field

- underflow would be when the negative exponent is too large to fit in the exponent field

(3pt) (1.6) Of all the available datapath components, please specify (place a checkmark) whether the component will be needed for the execution of instruction: `ld x1, 400(x2)`

Yes NO

☒ ☐ Instruction memory

☒ ☐ ALU

☒ ☐ Register file

☒ ☐ Immediate generation unit

☒ ☐ Data memory

(3pt) (1.7). For the following RISC-V instructions, they will use the arithmetic-logic unit (ALU) after reading the registers, please explain what is the function of ALU for each instruction.

|                                    | ALU function                  |
|------------------------------------|-------------------------------|
| a: <code>ADD x15, x16, x19</code>  | adding x16 with x19           |
| b: <code>SD x2, 200(x1)</code>     | sum of 200 offset to x1 the b |
| c: <code>BEQ x3, x4, offset</code> | subtracting x3 - x4           |

(30pts) 2.

(10 pts) (2.1) Given a 32-bit binary number, please demonstrate the range it could represent in 3 different cases.

(2 pts)(a) case 1: unsigned integer

$$[0, 2^{32}-1]$$

(2 pts) (b) case 2: signed integer

$$[-2^{31}, 2^{31}-1]$$

(6 pts) (c) case 3: single-precision IEEE 754 floating-point representation (please show the reasoning).

Smallest: 0000 0001  $\approx 1$

$$1 - 127 = -126$$

$$+ [1 \times 2^{-126} \approx 1.2 \times 10^{-38}]$$

largest

$$1111 1110 \approx 254$$

$$254 - 127 = 127$$

$$2 \times 2^{127} \approx 3.4 \times 10^{38}$$

(10 pts) (2.2) What is the decimal value of this IEEE 754 single-precision floating point

number sign exponent Fraction  
1 011 0001 1000 1110 111 0000 0000 0000

$$(-1)^{128} 64 32 16 8 4 2 1$$

$$0 1 1 0 0 0 1 1$$

$$64 + 32 + 2 + 1 = 99$$

$$99 - 127 = -28$$

$$-1.001 1101 1110 \times 2^{-28} \Rightarrow -1.23 \times 2^{-9}$$

$$256 208 + 14 = 4 - 0001 1101 1110$$

$$-5 \times 10^{-9}$$

(10pts) (2.3) Please represent the decimal number: -213.75 using single-precision floating point IEEE 754 standard.

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

$$213$$

signed  
1

exponent +  
1000 1000

Fraction  
1010 1011 1000  
0000 0000  
000

128 64 16 4 1  
1.101 0101

.50 .25  
.11

$$1.101010111 \times 2^7$$

$$E - 127 = 7$$

$$E = 134$$

$$1000 1000$$

$$1 1000 1000 1010 1011 1000 0000 0000 0000$$



x5 = 11

x6 = 9

(10 pts) 3. Overflow

Assume the registers x5 and x6 hold the values as follows:

x5: 0x B000 0000 0000 0000

x6: 0x 9000 0000 0000 0000

(3.1) (i) what is the value of x30 after execution of the following assembly code?

(ii) Is the result in x30 the desired result, or has there been overflow?

Add x30, x5, x6

i. x30 = 20

ii. desired result

(3.2) (i) what is the value of x30 after execution of the following assembly code?

(ii) Is the result in x30 the desired result, or has there been overflow?

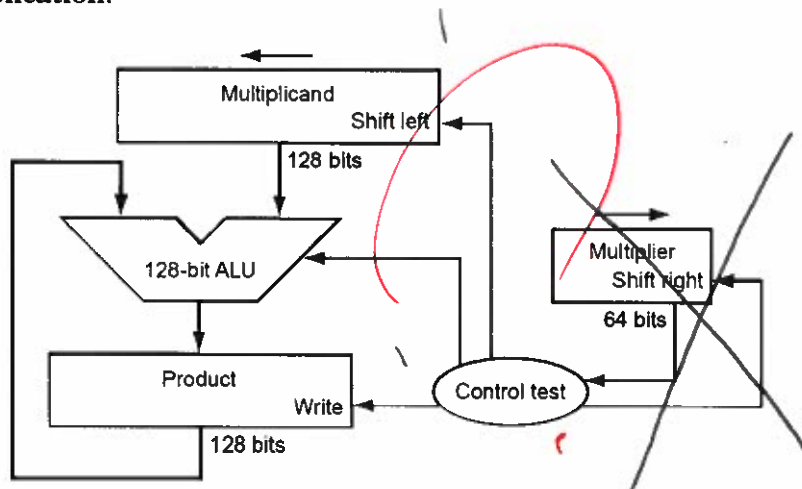
Sub x30, x5, x6

i. x30 = 2

ii. desired result

(10pts) 4. Integer multiplication. We will use 8-bit binary number as a case study

(5pts) (4.1) the following diagram demonstrate the first version of multiplication of two 64-bit binary number, Please revise it to perform the two 8-bit binary number multiplication.



(5pts) (4.2) Please briefly explain how many clock cycles do you need to perform a 8-bit multiplication following above diagram. Please explain the reasoning.

clock cycles =  $(3 \times 8) \times \# \text{ of occurrences}$   
3 stages      8 bit long

(19 pts) 5. Single cycle implementation – multi cycle implementation – pipelined implementation:

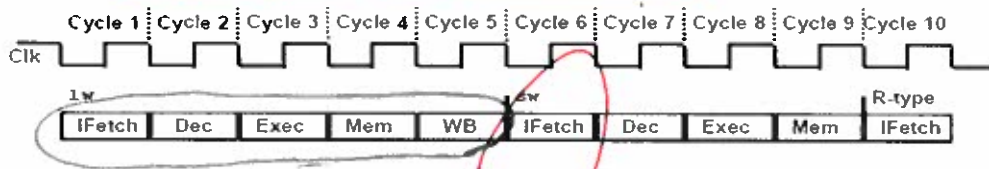
Calculate cycle time assuming negligible delays (for muxes, control unit, immediate Gen, PC access, shift left 1, wires) except: Instruction and Data Memory (200ps); ALU and adders (200ps); and Register File access (reads or writes) (100 ps)

- (5 pts) (5.1) (i) Please fill in the table and calculate the total time needed for each instruction  
(ii) What will be a good time for a single cycle time?

| Instruction Class    | Instruction memory | Register Read | ALU Operation | Data Mem | Register Write | Total |
|----------------------|--------------------|---------------|---------------|----------|----------------|-------|
| R type( such as SUB) | 200                | 100           | 200           | 0        | 100            | 600ps |
| Load (LW)            | 200                | 100           | 200           | 200      | 100            | 800ps |
| Store (SW)           | 200                | 100           | 200           | 200      | 0              | 700ps |
| BNE                  | 200                | 100           | 200           | 0        | 0              | 500p  |

ii. a good <sup>single</sup> cycle time will be 800ps

- (2 pts) (5.2) when we adopt a multicycle implementation. Each step will take 1 clock cycle, please explain what will be an appropriate cycle time?



5 clock cycle time

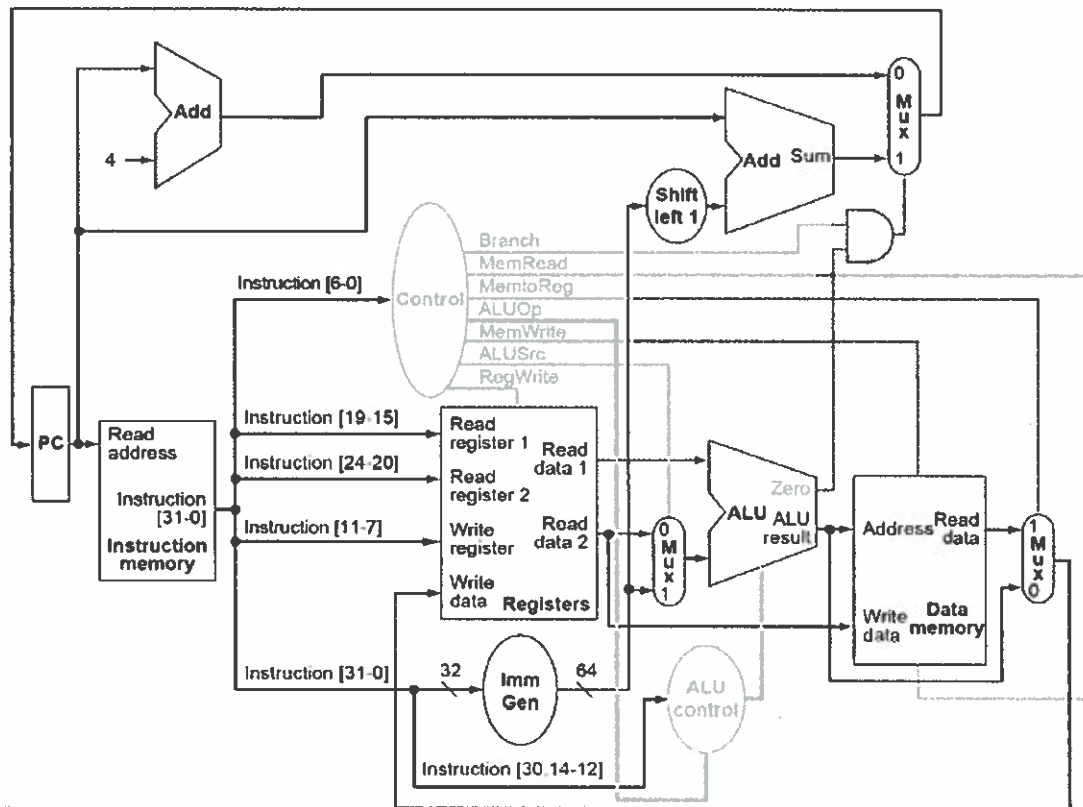
(4 pts) (5.3)

- (i) Please explain what is the pipelined implementation.

It is parallel execution of instructions that are back to back.

- (ii) If a new technology could shorten ALU operation by 20%, will it affect the speedup obtained from pipelining? Please briefly explain. It will not since the longest stage still has to be considered for all stages.
- (iii) What if the ALU operation now take 20% more time? Please briefly explain.

Yes since the ALU operation is now the longest stage, that will be the time considered for all stages increasing overall time.



(4 pts) 5.4. Please fill in the following table, which specify how to set the control signals when executing the 2 instructions.

| Instruction | Branch                           | MemtoReg                         | ALUSrc                           | RegWrite                         | MemRead                          | MemWrite                         | ALUOp |
|-------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|-------|
| SD          | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | 00    |
| ADDI        | <input checked="" type="radio"/> | <input type="radio"/>            | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | <input checked="" type="radio"/> | 01    |

(4 pts) 5.5. Please explain the detailed execution steps for the instruction:

ld x10, 100(x11)

- instructions are fetched from instruction memory
- register value reads from register file in this case x
- ALU computes sum of value offset from register file and extends the offset by 12 bits
- the sum of 100(x11) is used as an address for data memory
- the data from memory is written into x1 in register file.

(14 pts) 6. The following RISC-V code segment is corresponding to the C code:  
 $A = B + E; C = B - F;$

1. ld x10, 0(x6)  
 2. ld x11, 8(x6) 12  
 3. add x12, x10, x11 13  
 4. sd x12, 16(x6)  
 5. ld x13, 24(x6)  
 6. sub x15, x10, x13 13  
 7. sd x15, 40(x6)

(2 pts) 6.1 Please explain what is data hazard. Please give an example from the code sequence above.

A data hazard is a situation that prevents starting of the next instruction for the next cycle.

For line 2 and 3 since 3 is dependent on 2, 5 and 6 too

(4 pts) 6.2 Assume forwarding (aka bypassing) scheme is adopted here. (i) Please specify the data hazards even with the forwarding scheme and point out where stalls should be added. (ii) how many clock cycles do you need to implement the above code sequence assuming a 5-stage pipeline?

Data hazards for forwarding still include

i. that stalls can't be avoided if value is not yet computed when needed.

ii. 13 clock cycles  $5 + ((7-1) + 2) = 13$  <sup>stall</sup>

(4 pts) 6.3 (i) Please reorder the code segment (while do not change the function of the program) to avoid the stalls. (ii) how many clock cycles do you need to implement the above code sequence?

new order: 1, 2, 5, 3, 4, 6, 7.

ii. 11 clock cycles

(4 pts) 6.4 What is the minimum number of cycles needed to completely execute N instructions on a CPU with a M stage pipeline? Justify your formula.

$$\text{Clock Cycles} = M + (N - 1)$$

M is the set # of stages in the pipeline for the next instruction we would then take n-1, since a step has already begun due to being continuous

**BELOW AVERAGE**

**Below Average Student Original Graded Assignment**

**ELEG 4339: Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Exam 2

Spring 2022

Name: (Print) \_\_\_\_\_

*Torren Gray*

(Sign) \_\_\_\_\_

*Torren Gray*

|              |     |      |
|--------------|-----|------|
| 1            | 17  | 15.5 |
| 2            | 30  | 9    |
| 3            | 10  | 3.5  |
| 4            | 10  | 11.5 |
| 5            | 19  | 15.5 |
| 6            | 14  | 11   |
|              |     |      |
| <b>Total</b> | 100 | 56   |

Instructions:

You have 80 minutes to complete the exam. The exam totals 100 points; the points for each problem are indicated. This exam counts for 15% of your course grade.

**(17pts) 1. Short answer**

(2pt)(1.1). Which of the following belongs to **normalized scientific notation**.

- a.  $27.35 \times 10^9$       **(b)  $2.735 \times 10^9$**       c.  $0.2735 \times 10^9$

(2pt) (1.2). Is the IEEE 754 single-precision floating point number

1011 0001 1000 1110 1111 0000 0000 0000

- a. positive      **(b) negative**

(2pt) (1.3). Name the **special purpose register** that stores the **address** of the next instruction to be fetched from memory.

Answer: Program Counter (PC)

(2pt) (1.4). In the integer multiplication, if we ignore the sign bits, what is the length of the product of a **n-bit** multiplicand and an **m-bit** multiplier?

- a. n    b. m    c.  $2n$     d.  $2m$     **(e)  $n+m$**     f.  $n-m$     g. None of the above

(3pt) (1.5) What is overflow and underflow for floating point arithmetic?

*Overflow is when a positive exponent is too large to fit in the exponent field.*

*Underflow is when a negative exponent becomes too large to fit in the field.*

(3pt) (1.6) Of all the available datapath components, please specify (place a checkmark) whether the component will be needed for the execution of instruction: `ld x1, 400(x2)`

- |                                     |                                     |                           |
|-------------------------------------|-------------------------------------|---------------------------|
| Yes                                 | NO                                  |                           |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Instruction memory        |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | ALU                       |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Register file             |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | Immediate generation unit |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Data memory               |

(3pt) (1.7). For the following RISC-V instructions, they will use the arithmetic-logic unit (ALU) after reading the registers, please explain what is the function of ALU for each instruction.

|                                    | ALU function                                                                                                             |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| a: <code>ADD x15, x16, x19</code>  | <i>operates on <del>x16</del> x16 and x19 from register file<br/>copy portions of opcode to generate ALU fun</i>         |
| b: <code>SD x2, 200(x1)</code>     | <i>ALU computes the sum of value read from register <del>x1</del> x1 and the same is used for the address.</i>           |
| c: <code>BEQ x3, x4, offset</code> | <i>ALU compares the values obtained from the register value &amp; the sign extends 12 bits of the instruction offset</i> |



9

(30pts) 2.

(10 pts) (2.1) Given a 32-bit binary number, please demonstrate the range it could represent in 3 different cases.

✓ (2 pts)(a) case 1: unsigned integer

$$[0, 2^{32}-1]$$

✓ (2 pts) (b) case 2: signed integer

$$[-2^{31}, 2^{31}-1]$$

0 (6 pts) (c) case 3: single-precision IEEE 754 floating-Point representation (please show the reasoning).

$$-126 \text{ to } 127$$



3

(10 pts) (2.2) What is the decimal value of this IEEE 754 single-precision floating point number

1011 0001 1000 1110 1111 0000 0000 0000

$$99 - 127 = -28$$

$$2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-9} + 2^{-10} + 2^{-11} = x$$

$$x \cdot 2^{28} =$$

✓ (10pts) (2.3) Please represent the decimal number: -213.75 using single-precision floating point IEEE 754 standard.

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

128  
64  
16  
-4  
-1

11010101

$$E = 127 = 2$$

$$= 125$$

$$.75 \times 2 = 1.5$$

$$.5 \times 2 = 1.0$$

$$.0 \times 2 = 0.0$$

✓ 1 011 1101 11010101 110 0000000000000000

3

**(10 pts) 3. Overflow**

Assume the registers x5 and x6 hold the values as follows:

x5: 0x B000 0000 0000 0000

x6: 0x 9000 0000 0000 0000

- (3.1) (i) what is the value of x30 after execution of the following assembly code?  
(ii) Is the result in x30 the desired result, or has there been overflow?

Add x30, x5, x6

i - 20  
ii - no overflow

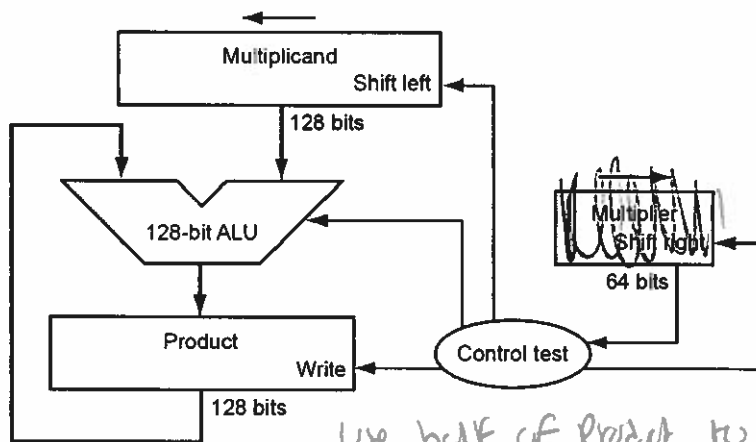
- (3.2) (i) what is the value of x30 after execution of the following assembly code?  
(ii) Is the result in x30 the desired result, or has there been overflow?

Sub x30, x5, x6

i - 2  
ii - no overflow

**(10pts) 4. Integer multiplication. We will use 8-bit binary number as a case study**

(5pts) (4.1) the following diagram demonstrate the first version of multiplication of two 64-bit binary number, **Please revise it to perform the two 8-bit binary number multiplication.**



Use half of Product to shift multiplier right

(5pts) (4.2) Please briefly explain how many clock cycles do you need to perform a 8-bit multiplication following above diagram. Please explain the reasoning.

Num of cycles = (num of steps) + 4 + (num of shifts)  
= 8 + 4  
= 12 cycles

15/15

(19 pts) 5. Single cycle implementation – multi cycle implementation – pipelined implementation:

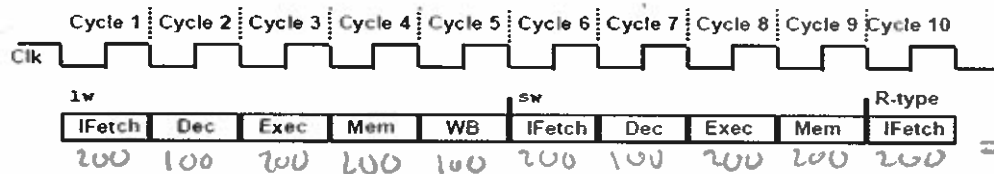
Calculate cycle time assuming negligible delays (for muxes, control unit, immediate Gen, PC access, shift left 1, wires) except: Instruction and Data Memory (200ps); ALU and adders (200ps); and Register File access (reads or writes) (100 ps)

$$200 + 100 + 200 + 200 + 100 = 800$$

- (5 pts) (5.1) (i) Please fill in the table and calculate the total time needed for each instruction  
(ii) What will be a good time for a single cycle time?

| Instruction Class    | Instruction memory | Register Read | ALU Operation | Data Mem | Register Write | Total |
|----------------------|--------------------|---------------|---------------|----------|----------------|-------|
| R type( such as SUB) | 200                | 100           | 200           |          | 100            | 600   |
| Load (LW)            | 200                | 100           | 200           | 200      | 100            | 800   |
| Store (SW)           | 200                | 100           | 200           | 200      |                | 700   |
| BNE                  | 200                | 100           | 200           |          |                | 500   |

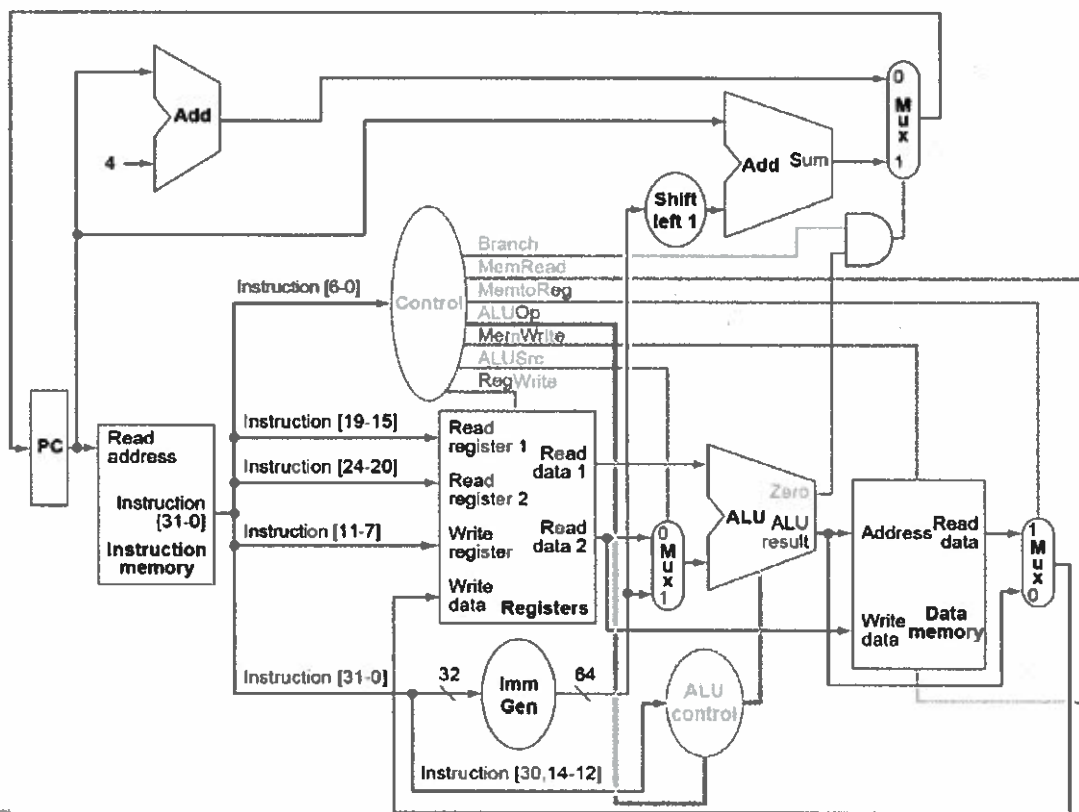
- (2 pts) (5.2) when we adopt a multicycle implementation. Each step will take 1 clock cycle, please explain what will be an appropriate cycle time?



$$= \frac{1700}{10} = 170$$

(4 pts) (5.3)

- (i) Please explain what is the pipelined implementation.  
*running programs by overlapping execution of instructions*
- (ii) If a new technology could shorten ALU operation by 20%, will it affect the speedup obtained from pipelining? Please briefly explain. *yes because it'll take less time to execute other quicker operations*
- (iii) What if the ALU operation now take 20% more time? Please briefly explain.  
*This will affect the speedup because the operations could be delayed*



(4 pts) 5.4. Please fill in the following table, which specify how to set the control signals when executing the 2 instructions.

| Instruction | Branch | MemtoReg | ALUSrc | RegWrite | MemRead | MemWrite | ALUOp |
|-------------|--------|----------|--------|----------|---------|----------|-------|
| SD          | 0      | X        | 1      | 1        | 0       | 1        | 00    |
| ADDI        | 0      | 0        | 0      | 1        | 0       | 0        | 01    |

(4 pts) 5.5. Please explain the detailed execution steps for the instruction:

ld x10, 100(x11)

Instruction fetched from Instruction memory + PC is incremented

Register value (x11) is read from register file

ALU computes the sum of value read from register value + 100

Sum of the ALU is used as address for data memory  
data from memory unit is written to register (x10)

(14 pts) 6. The following RISC-V code segment is corresponding to the C code:  
 $A = B + E$ ;  $C = B - F$ ;

1. ld x10, 0(x6)
2. ld x11, 8(x6)
3. add x12, x10, x11
4. sd x12, 16(x6)
5. ld x13, 24(x6)
6. sub x15, x10, x13
7. sd x15, 40(x6)

(2 pts) 6.1 Please explain what is data hazard. Please give an example from the code sequence above.

Need to wait for previous instruction to complete its data read/write  
 12 x11 8(x6)  
 add x12 x10 x11 ✓

(4 pts) 6.2 Assume forwarding (aka bypassing) scheme is adopted here. (i) Please specify the data hazards even with the forwarding scheme and point out where stalls should be added. (ii) how many clock cycles do you need to implement the above code sequence assuming a 5-stage pipeline?

stall add x12, x10, x11  
 stall sub x15, x10, x13

$$CC = 7 + 5 + 2 = 14$$

(4pts) 6.3 (i) Please reorder the code segment (while do not change the function of the program) to avoid the stalls. (ii) how many clock cycles do you need to implement the above code sequence?



$$CC = 7 + 5 + 0 = 12$$

(4pts) 6.4 What is the minimum number of cycles needed to completely execute N instructions on a CPU with a M stage pipeline? Justify your formula.

$$N + M$$

7 instructions  
 5 stage pipeline  
 0 stalls

$$= 12$$

# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**SPRING 2022**

**Title of Assignment: Project 1**

**PRAIRIE VIEW A&M UNIVERSITY**  
**COLLEGE OF ENGINEERING**  
**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**  
**ELEG 4339 Computer Organization and Design**

Project 1 Venus Simulator

This computer organization class is based on RISC-V assembly language. There are simulators for practicing. This project is based on one of the simulators, Venus. Venus is an open-source software developed by UC Berkeley. GitHub can be found at <https://github.com/kvakil/venus> with more features to explore. This project is based on its online simulator (<https://www.kvakil.me/venus/>).

Note: If you encounter with access problem to above website, you can use <https://venus.cs61c.org/> instead. Interface is a little different, but all our practices can be implemented the same way.

The goals of this project is: 1) Get familiar with Venus simulator; 2) Observe how registers and memory support computing; and 3) Review RISC-V assembly programming.

There are three modules in this project to achieve the above goals.

### Learning Module 1: Introduction of Venus Online Simulator

Step1: Go to website <https://www.kvakil.me/venus/>, you can see the interface of an “Editor” and a “Simulator.” Editor is for you to enter your program. Simulator includes machine code, registers, memory, console output, as well as debug actions: Run, Step, Rev, Reset, and Dump.

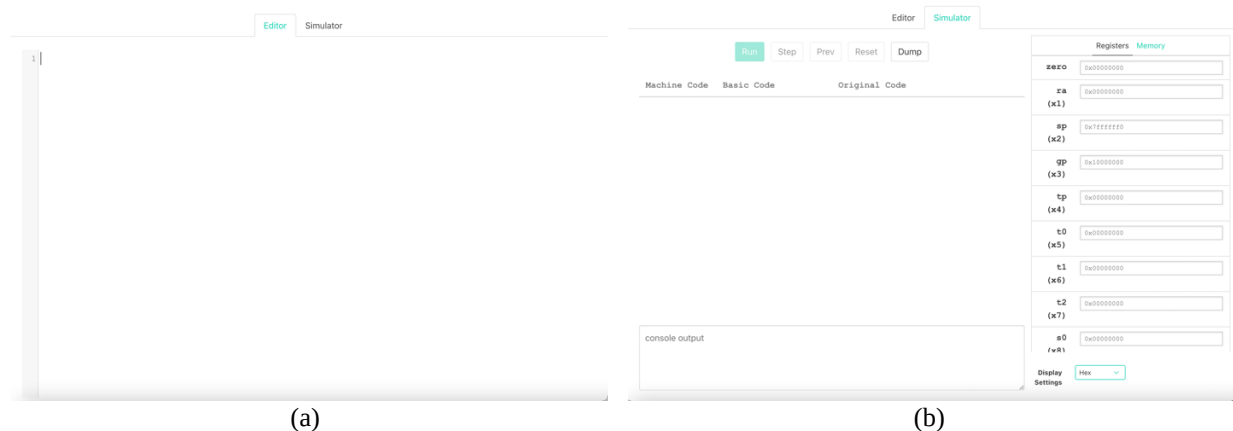


Figure 1: Venus interface

Registers: There are 32 32-bit registers supported by RISC-V architecture to hold operands. They are numbered from zero, then x1 to x31. Some are named for their special purposes.

Memory: Memory in the simulator is organized as 4 bytes per group with an entry address shown on the left. Machine code is written at the beginning of the memory location as 4 bytes per instruction. (Review RISC-V each machine language is of 32 bits.) User can jump to other memory segments like “Data” “Heap” or “Stack”.

Display setting is “Hex” by default.

Debug: The simulator provides run the whole program (Run), single step (Step), go to previous instruction (Prev), go to the beginning of the program (Reset), and copy machine code to clipboard (Dump).

Step 2: First program.

- Type “add x9, x20, x21” in Editor field.

- Put 1 as initial value of x20 and 2 as initial value of x21.

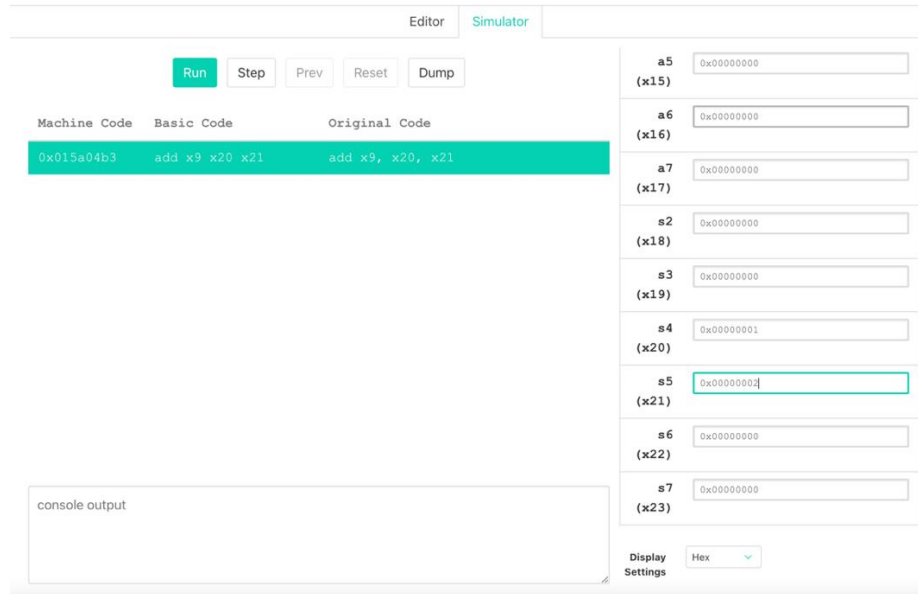


Figure 2: Simple add instruction on textbook page 87

- Click “Run” to observe result in x9.

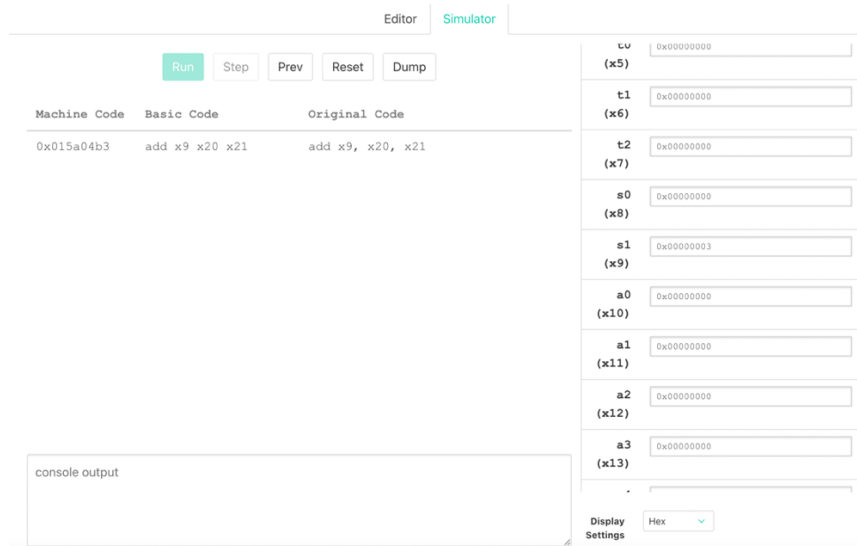


Figure 3: Observe register changes after “Run”

- Click “Reset” to see all registers go back to zeros.

### Step 3: Analyze machine code.

- Switch to “Memory”. Observe the machine code stored at the beginning of the memory.  
0x00000000      b3    04    5a    01
- Click “Dump”. The machine code is copied in clipboard and shown in output.
- What is the endian choice of RISC-V? Verify with the binary code as shown in textbook page 88.

### Learning Module 2: Register and memory access



Case 1: Operands from registers.

Step 1: Give initial values to registers: x1 = 3, x2 = 4.

Step 2: Type “ADD x3, x2, x1” in Editor.

Step 3: Click “Run”. Get a screenshot from the interface to show the result in x3.

Step 4: Work out machine code based on what you learned from Chapter 2. Analyze and verify the machine code obtained from the simulator.

Case 2: Store value in memory.

Step 1: Give initial value x10 = 4.

Step 2: Type following code to “Editor”.

```
lw      x9, 120(x10)
add     x9, x21, x9
addi    x9, x9, 1
sw      x9, 120(x10)
```

Step 3: Simulate the code using single steps. Observe the changes to registers. At last, verify the final memory location value and show the calculation of memory address as the source and destination of lw and sw.

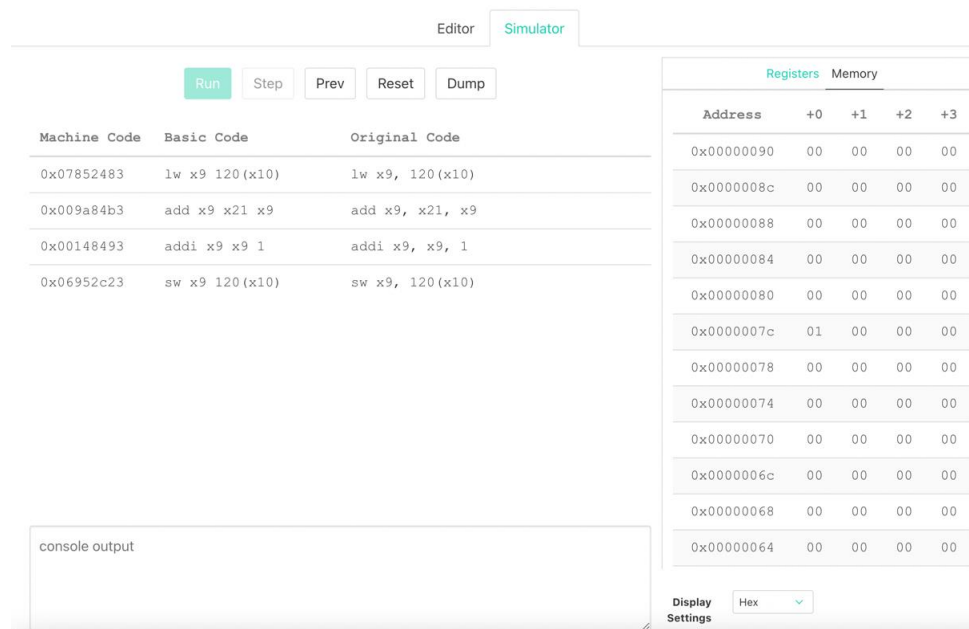


Figure 4: Memory

Step 4: (Optional) Change the initial value of x10 to 2, 3, or 8, re-do the above procedures, and explain each case how the memory locations are calculated. Verify it with simulation results.

### Learning Module 3: Control statement

Step 1: Give initial value x1 = 14.

Step 2: Type below code in “Editor”.

```
addi x2, x0, 1
loop:
    sub x1, x1, x2
    sw x1, 40(x1)
    blt x0, x1, loop
```

Step 3: Use single step to see the changes in registers and memory locations. One example output during the procedure is illustrated in Figure 5.

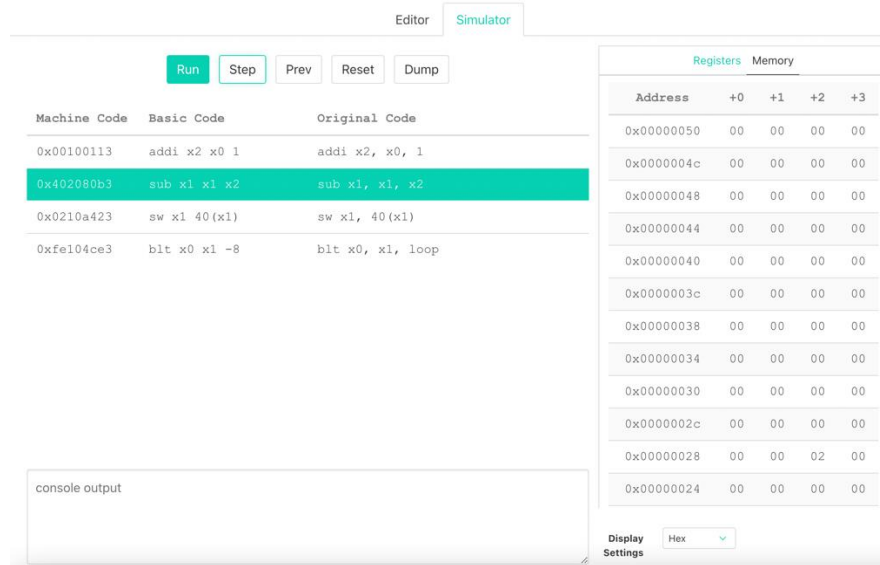


Figure 5: Sample output

Step 4: Based on your understanding of the code, write a C code program and draw flowchart to explain the control structure.

#### Project Question:

1. Write RISC-V assembly language to implement the common function  $c = \max(a, b)$  which assigns to  $c$  the value of  $a$  or  $b$  which is larger. You can choose either registers or memory locations to assist the action.

Briefly describe the problem. Provide two screenshots: One before program run with initialized register values or memory values. One after program run with register or memory location values. Attach assembly code and explain two lines of machine codes from your program in binary level.

2. Write RISC-V assembly language to calculate the function of square sum. Put an integer  $n$  into register  $a0$ . Calculate  $n^2 + (n-1)^2 + (n-2)^2 + \dots + 1^2$ , and put the result in  $a0$ .

Provide two screenshots: One before program run with initialized register values. One after program run with register values. Attach assembly code and explain how the codes work.

#### Project Requirements:

This is a mandatory assignment and will take 10% of your final grade. Maximum of two students can work on one project. They can share the same report, but each student must upload the report to canvas individually before the deadline (April 5 before midnight).

Each report will contain:

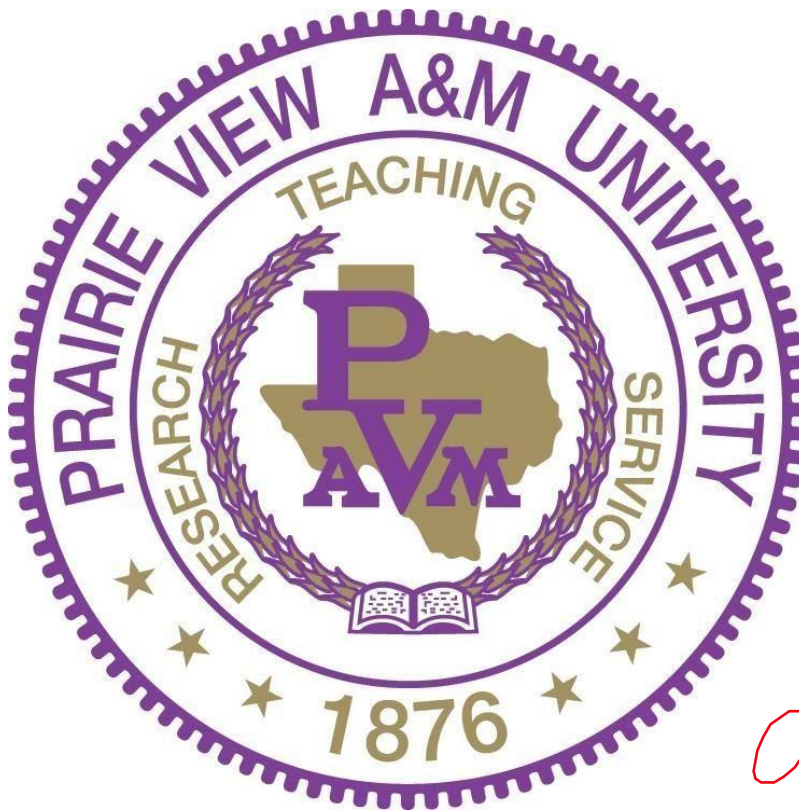
1. Cover page with course, assignment, and student information. (0.5 points)
2. Goals and procedures. (0.5 points)

3. Answer questions from:
  - a. Learning Module 1 Step 3 Analyze machine code; (1 points)
  - b. Learning Module 2 Case 1 Step 4; (1 points)
  - c. Learning Module 2 Case 2 Step 3 and optional Step 4; (1 points)
  - d. Learning Module 3 Step 4 (1 points)
4. Reply two project questions with screen shots, narrations, Assembly source codes and etc. (4.5 points)
5. Conclusions. (0.5 points)

**ABOVE AVERAGE**

**Above Average Student Original Graded Assignment**

**PRAIRIE VIEW A&M UNIVERSITY**  
**ROY G. PERRY COLLEGE OF ENGINEERING**



93

***ELEG 4339-P01 Computer Organization and Design***

**Course Topic: Project 1 Venus Simulator**

**Instructor: Dr. Xiangfang & Dr.Cui**

**Submitted by: John Barrett & Abel Mendoza**

**Date: (03/29/2022)**

## Goals and Procedure:

Goals- The goal of this project is to get acquainted with Venus online RISC-V simulation software. We must observe how registers and memory support computing as well as familiarizing ourselves with the way RISC-V assembly code is written.

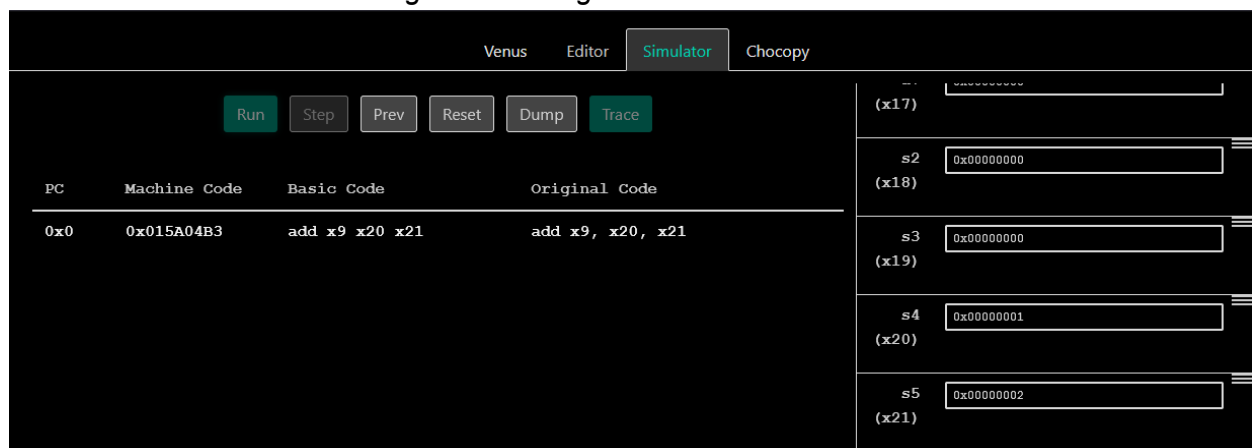
Procedures-

- 1) Each member of the group worked through the learning modules to acquaint themselves with the simulation software and RISC-V assembly code.
- 2) Each member then tackled one of the project questions.
- 3) Then each member reviewed the other member's solutions.

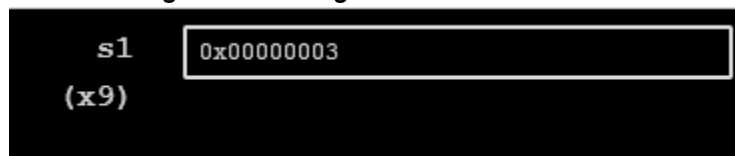
## Results: Learning Modules 1-3

*Learning Module 1: Introduction of Venus Online Simulator:*

*Image 1: Learning Module 1 Part 2 Before*



*Image 2: Learning Module 1 Part 2 After*



*Image 3: Learning Module 1 Part 3 (Little Endian)*



Machine code: 0x015A04B3

We can see that the bytes are stored as least significant first and most significant last; which is called "Little Endian."

## Learning Module 2: Register and memory access:

Image 4: Module 2 Case 1 Step 3

Run Step Prev Reset Dump Trace

| PC  | Machine Code | Basic Code   | Original Code |
|-----|--------------|--------------|---------------|
| 0x0 | 0x001101B3   | add x3 x2 x1 | add x3,x2,x1  |

Registers Memory Cache VDB

Integer (R) Floating (F)

zero 0x00000000

ra (x1) 0x00000003

sp (x2) 0x00000004

gp (x3) 0x00000007

Machine Code: 0x001101B3 = 0b0000,0000,0010,0010,0000,0001,1011,0011

ADD is a R-type instruction so it is broken up as follows

0000000,00001,00010,000,00011,0110011

funct7, rs2, rs1, funct3, rd, Opcode

The ADD command has the following Opcode, function 3, and function 7.

0110011, 000, 0000000

rs2 = 00001 => 1

rs1 = 00010 => 2

rd = 00011 => 3

Thus, we can see that it matches what the code intends to do.

→ 0 2

Image 5: Module 2 Case 2 Step 3 (Memory)

Run Step Prev Reset Dump Trace

| PC  | Machine Code | Basic Code     | Original Code   |
|-----|--------------|----------------|-----------------|
| 0x0 | 0x07852483   | lw x9 120(x10) | lw x9, 120(x10) |
| 0x4 | 0x009A84B3   | add x9 x21 x9  | add x9, x21, x9 |
| 0x8 | 0x00148493   | addi x9 x9 1   | addi x9, x9, 1  |
| 0xc | 0x06952C23   | sw x9 120(x10) | sw x9, 120(x10) |

Registers Memory Cache VDB

| Address    | +0 | +1 | +2 | +3 |
|------------|----|----|----|----|
| 0x00000090 | 00 | 00 | 00 | 00 |
| 0x0000008C | 00 | 00 | 00 | 00 |
| 0x00000088 | 00 | 00 | 00 | 00 |
| 0x00000084 | 00 | 00 | 00 | 00 |
| 0x00000080 | 00 | 00 | 00 | 00 |
| 0x0000007C | 01 | 00 | 00 | 00 |

→ 05

✓



Image 6: Module 2 Case 2 Step 3 (Registers)

The screenshot shows a debugger window with a table of assembly instructions and a register window on the right.

| PC  | Machine Code | Basic Code     | Original Code   |
|-----|--------------|----------------|-----------------|
| 0x0 | 0x07852483   | lw x9 120(x10) | lw x9, 120(x10) |
| 0x4 | 0x009A84B3   | add x9 x21 x9  | add x9, x21, x9 |
| 0x8 | 0x00148493   | addi x9 x9 1   | addi x9, x9, 1  |
| 0xc | 0x06952C23   | sw x9 120(x10) | sw x9, 120(x10) |

Register window (x7):

- s0 (x8): 0x00000000
- s1 (x9): 0x00000001
- a0 (x10): 0x00000004

Learning Module 3: Control statement:

Image 7: Learning Module 3 Step 3 after a couple of step throughs

The screenshot shows a debugger window with a table of assembly instructions and a register window on the right. The instruction at PC 0x4 is highlighted in green.

| PC  | Machine Code | Basic Code   | Original Code  |
|-----|--------------|--------------|----------------|
| 0x0 | 0x00100113   | addi x2 x0 1 | addi x2,x0,1   |
| 0x4 | 0x402080B3   | sub x1 x1 x2 | sub x1,x1,x2   |
| 0x8 | 0x0210A423   | sw x1 40(x1) | sw x1,40(x1)   |
| 0xc | 0xFE104CE3   | blt x0 x1 -8 | blt x0,x1,loop |

Register window:

- zero: 0x00000000
- ra (x1): 0x00000007
- sp (x2): 0x00000001

Image 8: Learning Module 3 Step 3 after even more step throughs

The screenshot shows a debugger window with a table of assembly instructions and a register window on the right. The instruction at PC 0xc is highlighted in green.

| PC  | Machine Code | Basic Code   | Original Code  |
|-----|--------------|--------------|----------------|
| 0x0 | 0x00100113   | addi x2 x0 1 | addi x2,x0,1   |
| 0x4 | 0x402080B3   | sub x1 x1 x2 | sub x1,x1,x2   |
| 0x8 | 0x0210A423   | sw x1 40(x1) | sw x1,40(x1)   |
| 0xc | 0xFE104CE3   | blt x0 x1 -8 | blt x0,x1,loop |

Register window:

- zero: 0x00000000
- ra (x1): 0x00000005
- sp (x2): 0x00000001

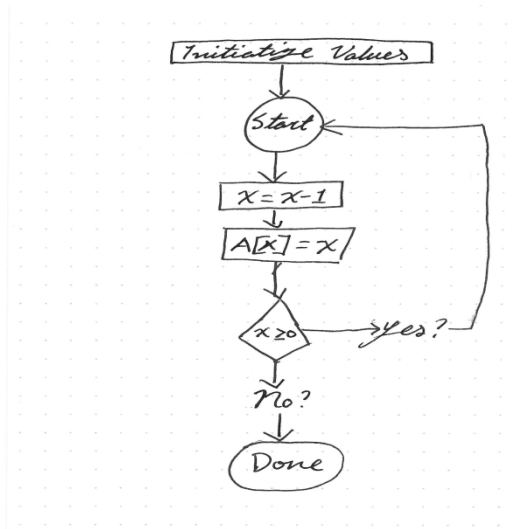
C Code-

```

Int x = 14;
Int A[x];
while(x >= 0){
    x--;
    A[x] = x; }

```

Flow Chart-



## Project Question: 1 & 2

### Question 1:

Write RISC-V assembly language to implement the common function  $c = \max(a,b)$  which assigns to  $c$  the value of  $a$  or  $b$  which is larger. You can choose either registers or memory locations to assist the action.

Images & corresponding descriptions-

**Screenshot #1: Before the program ran with initialized register values or memory values.**

| PC  | Machine Code | Basic Code     | Original Code            |
|-----|--------------|----------------|--------------------------|
| 0x0 | 0x016AD663   | bge x21 x22 12 | bge x21,x22,a_is_greater |
| 0x4 | 0x000B0A33   | add x20 x22 x0 | add x20,x22,x0           |
| 0x8 | 0x016A0463   | beq x20 x22 8  | beq x20,x22,b_is_best    |
| 0xc | 0x000A8A33   | add x20 x21 x0 | add x20,x21,x0           |

(x17)

s2  
(x18) 0x00000000

s3  
(x19) 0x00000000

s4  
(x20) 0x00000000

s5  
(x21) 0x00000004

s6  
(x22) 0x00000006

**Screenshot #2: After the program runs with register or memory location values.**

| PC  | Machine Code | Basic Code     | Original Code            |
|-----|--------------|----------------|--------------------------|
| 0x0 | 0x016AD663   | bge x21 x22 12 | bge x21,x22,a_is_greater |
| 0x4 | 0x000B0A33   | add x20 x22 x0 | add x20,x22,x0           |
| 0x8 | 0x016A0463   | beq x20 x22 8  | beq x20,x22,b_is_best    |
| 0xc | 0x000A8A33   | add x20 x21 x0 | add x20,x21,x0           |

|          |            |
|----------|------------|
| (x17)    |            |
| s2 (x18) | 0x00000000 |
| s3 (x19) | 0x00000000 |
| s4 (x20) | 0x00000006 |
| s5 (x21) | 0x00000004 |
| s6 (x22) | 0x00000006 |

Attach assembly code and explain two lines of machine codes from your program in binary level.

#### Assembly Code-

```

bge x21,x22,a_is_greater
add x20,x22,x0
beq x20,x22,b_is_best
a_is_greater:
add x20,x21,x0
b_is_best:

```

#### Machine Code- Hex

```

0x016AD663
0x000B0A33
0x016A0463
0x000A8A33

```

#### Machine Code- Binary

```

00000001011010101101011001100011
000000000000010110000101000110011
0000000101101010100000010001100011
000000000000010101000101000110011

```

#### Analysis of lines 2 and 4:

##### Line 2-

0000000,00000,10110,000,10100,0110011

Opcode = 0110011

Function3 = 000

Function7 = 0000000

The command that has these 3 attributes is the ADD command

Knowing that it is an ADD command we can read what registers it is using.

Rd = 10100 = x20

Rs1 = 10110 = x22

Rs2 = 00000 = x0

So this is adding the contents of register x22 to that of x0 and storing it in x20.

Line 4-

0000000,00000,10101,000,10100,0110011

Opcode = 0110011

Function3 = 000

Function7 = 0000000

The command that has these 3 attributes is the ADD command

Knowing that it is an ADD command we can read what registers it is using.

Rd = 10100 = x20

Rs1 = 10101 = x21

Rs2 = 00000 = x0

So this is adding the contents of register x21 to that of x0 and storing it in x20.

## Question 2:

Write RISC-V assembly language to calculate the function of square sum. Put an integer n into register a0. Calculate  $n^2 + (n-1)^2 + (n-2)^2 + \dots + 1^2$ , and put the result in a0.

**Screenshot #1: Before the program ran with initialized register values.**

|                                           |              |                 |                 |             |            |
|-------------------------------------------|--------------|-----------------|-----------------|-------------|------------|
| <div>Run Step Prev Reset Dump Trace</div> |              |                 |                 | (x19)       |            |
| PC                                        | Machine Code | Basic Code      | Original Code   | s4<br>(x20) | 0x00000000 |
| 0x0                                       | 0x00100113   | addi x2 x0 1    | addi x2,x0,1    | s5<br>(x21) | 0x00000000 |
| 0x4                                       | 0x036B0C33   | mul x24 x22 x22 | mul x24,x22,x22 | s6<br>(x22) | 0x000001F4 |
| 0x8                                       | 0x018A0A33   | add x20 x20 x24 | add x20,x20,x24 | s7<br>(x23) | 0x00000000 |
| 0xc                                       | 0x402B0B33   | sub x22 x22 x2  | sub x22,x22,x2  |             |            |
| 0x10                                      | 0xFE0B1AE3   | bne x22 x0 -12  | bne x22,x0,loop |             |            |

**Screenshot #2 After the program ran with register values.**

| <div> <div>Run</div> <div>Step</div> <div>Prev</div> <div>Reset</div> <div>Dump</div> <div>Trace</div> </div> |              |                 |                 |  |  |
|---------------------------------------------------------------------------------------------------------------|--------------|-----------------|-----------------|--|--|
| PC                                                                                                            | Machine Code | Basic Code      | Original Code   |  |  |
| 0x0                                                                                                           | 0x00100113   | addi x2 x0 1    | addi x2,x0,1    |  |  |
| 0x4                                                                                                           | 0x036B0C33   | mul x24 x22 x22 | mul x24,x22,x22 |  |  |
| 0x8                                                                                                           | 0x018A0A33   | add x20 x20 x24 | add x20,x20,x24 |  |  |
| 0xc                                                                                                           | 0x402B0B33   | sub x22 x22 x2  | sub x22,x22,x2  |  |  |
| 0x10                                                                                                          | 0xFE0B1AE3   | bne x22 x0 -12  | bne x22,x0,loop |  |  |

(x19)

s4 (x20) 0x027DB106

s5 (x21) 0x00000000

s6 (x22) 0x00000000

s7 (x23) 0x00000000

s8 (x24) 0x00000001

### Assembly code attachment:

```
addi x2,x0,1
```

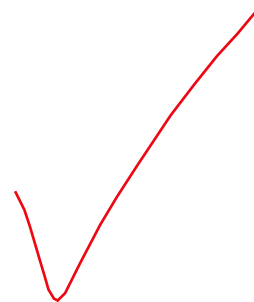
```
Loop:
```

```
mul x24,x22,x22
```

```
add x20,x20,x24
```

```
sub x22,x22,x2
```

```
bne x22,x0,loop
```



\*Note this code only works if the Sum of Squares is 8 Hexadecimal digits or less.

### Explanation on how the codes work:

The code starts by setting up a register with the value of 1. This could also be forgone and ADDI with negative 1 used instead. Then the code sets up the loop that is used to perform the summation. The original value stored in x22 is then multiplied by itself to get the squared value and stored in an intermediary register (x24). Then the summation step where the value in x24 is added to the sum in register x20. Then the number is stepped down by one so that the next term is set up to be squared. Then the loop decides if it needs to loop by testing if it has reached the last term or not (when x22 = 0). If it has not, it loops and the steps are done again.

### Conclusion:

In this project, we learned and interacted with one of the simulators for RISC-V assembly language. We used an open-source simulator named Venus. From here we are able to analyze how registers and memory interact with computing and code assembly. By interacting with the venus interface we were able to have a better understanding of how the 32-bit registers supported by the RISC-V architecture hold the necessary operands. With the provided interface we were able to use debugging commands such as Run, Step, Prev, Reset, and (Dump) which was necessary to copy machine code to the clipboard. This project helped us reinforce and apply concepts that have been taught so far in computer organization and design. Therefore, by putting our conceptual understanding to this simulation we were able to successfully complete the project.

**AVERAGE**

## **Average Student Original Graded Assignment**



**ELEG 4339**

**Computer Organization and Architecture:**

**Project** 1



8.5

**Xavier Dukes**

**04/05/2022**

## 2. Goals and procedures.

### Goals:

The goal of this project is for students to gain experience using the Venus simulation software and visualizing the affects of assembly code on registers and memory values in real time. Students will use their knowledge of functions, binary conversions, instructions and addressing to visualize how and where results of RISC-V operations are stored.

### Procedures:

Opening Venus simulation software and entering assembly code into text editor. For each complete assembly code program students should observe the changes in the registers or in memory locations if applicable and verify that the appropriate results are stored where they should be. During the validation of the results the machine code of the instruction should be checked as well.

## 3. Answer questions from:

### a. Learning Module 1 Step 3 Analyze machine code:

- The Machine code is represented by the Hex value 0x015A04B3 with B3 being the Opcode for the add function as the last 7 bits are 0110011
- It is stored as 0x00000000 b3 04 5a 01
- From the Screen Shot shown in image B, we can see the least significant byte is stored first in memory as B3 is stored first
- Making the RISC-V architecture Little Endianness

### b. Learning Module 2 Case 1 Step 4:

- Machine Code:
  - 0000 0000 0001 0001 0000 0001 1011 0011
  - 0x001101B3 as HEX

This is the correct machine code as the first 7 bits are for the funct7 field, while the next 5 bits and the proceeding 5 bits after that represent the two registers that hold the values for the

operation (in light blue and green). Highlighted in dark blue is the 3-bit funct3 field with the 5-bit field for the results register represented behind it (in red) and finally the 7-bit opcode for the ADD function.

c. Learning Module 3 Step 4:

```
addi x2, x0, 1
loop:
    sub x1, x1, x2
    sw x1, 40(x1)
    blt x0, x1, loop
```

C++ Program:

```
Int Num1 = 14;
Int num2 = 1;
While ( 0 < num1)
{
    Num1 = num1 - num2;
}
```

This C++ program initializes two values as done in the assembly code in registers x1 and x2 with 14 and 1 respectively. The program then enters a loop and executes the operation of subtracting 1 until the value in register x1 or num1 in this case is equal to zero at which point the loop ends as the condition is no longer true. In the assembly program the branching into the loop statement continues as long as the value 0 in reg x0 is less than the value in x1 during each iteration of the loop.

flowchart? -1.5

First Program:

Results:

The screenshot shows the Venus simulator interface. The top navigation bar includes tabs for Venus, Editor, Simulator, and Chocopy. The Simulator tab is active, displaying a table of execution data and a list of registers.

| PC  | Machine Code | Basic Code     | Original Code    |
|-----|--------------|----------------|------------------|
| 0x0 | 0x015A04B3   | add x9 x20 x21 | add x9, x20, x21 |

Registers (x4 to x12):

- tp (x4): 0
- t0 (x5): 0
- t1 (x6): 0
- t2 (x7): 0
- s0 (x8): 0
- s1 (x9): 3
- a0 (x10): 0
- a1 (x11): 0
- a2 (x12): 0

Display Settings: Decimal

Image A

The screenshot shows the Venus simulator interface. The top navigation bar includes tabs for Venus, Editor, Simulator, and Chocopy. The Simulator tab is active, displaying a table of execution data and a list of registers.

| PC  | Machine Code | Basic Code     | Original Code    |
|-----|--------------|----------------|------------------|
| 0x0 | 0x015A04B3   | add x9 x20 x21 | add x9, x20, x21 |

Registers (x4 to x12):

- tp (x4): 0
- t0 (x5): 0
- t1 (x6): 0
- t2 (x7): 0
- s0 (x8): 0
- s1 (x9): B3
- a0 (x10): 0
- a1 (x11): 0
- a2 (x12): 0

Display Settings: Hex

Image B

## Discussion:

We can see the machine code generated and its format when using register operands for an operation. As well as uncovering the Endianness type of the RISC-V architecture. These operations used values stored in registers in order to execute a give operational code.

## Second Program:

## Results:

The screenshot displays the Venus RISC-V simulator interface. The main window is divided into several sections:

- Top Bar:** Contains tabs for "Venus", "Editor", "Simulator" (active), and "Chocopy".
- Control Buttons:** A row of buttons including "Run", "Step", "Prev", "Reset", "Dump", and "Trace".
- Instruction Table:** A table showing the current instruction being executed.

| PC  | Machine Code | Basic Code   | Original Code  |
|-----|--------------|--------------|----------------|
| 0x0 | 0x001101B3   | add x3 x2 x1 | ADD x3, x2, x1 |
- Registers Panel:** A list of registers on the right side, categorized into "Integer (R)" and "Floating (F)". The visible registers and their values are:
  - zero: 0
  - ra (x1): 3
  - sp (x2): 4
  - gp (x3): 7
  - tp (x4): 0
  - t0 (x5): 0
  - t1 (x6): 0
  - t2 (x7): 0
  - s0 (x8): 0
- Console Output:** A text area at the bottom left labeled "console output" with buttons for "Copy!", "Download!", and "Clear!".
- Display Settings:** A dropdown menu set to "Decimal".

The Windows taskbar at the bottom shows the system clock as 4:05 PM on 3/26/2022, with a temperature of 85°F.

## Image C

The screenshot shows the Venus simulator interface. The top navigation bar includes tabs for Venus, Editor, Simulator (active), and Chocopy. Below this, there are control buttons: Run, Step, Prev, Reset, Dump, and Trace. The main display area is divided into four columns: PC, Machine Code, Basic Code, and Original Code. The PC value is 0x0, the Machine Code is 0x001101B3, the Basic Code is `add x3 x2 x1`, and the Original Code is `ADD x3, x2, x1`. At the bottom left, there is a text box containing the value 0x001101B3. On the right side, there is a table showing the state of registers, memory, cache, and VDB. The table has columns for Address, +0, +1, +2, and +3. The registers section shows the following values:

| Address    | +0 | +1 | +2 | +3 |
|------------|----|----|----|----|
| 0x00000018 | 00 | 00 | 00 | 00 |
| 0x00000014 | 00 | 00 | 00 | 00 |
| 0x00000010 | 00 | 00 | 00 | 00 |
| 0x0000000C | 00 | 00 | 00 | 00 |
| 0x00000008 | 00 | 00 | 00 | 00 |
| 0x00000004 | 00 | 00 | 00 | 00 |
| 0x00000000 | B3 | 01 | 11 | 00 |
| -----      | -- | -- | -- | -- |
| -----      | -- | -- | -- | -- |
| -----      | -- | -- | -- | -- |
| -----      | -- | -- | -- | -- |

At the bottom right, there is a 'Display Settings' dropdown menu set to 'Hex'. The bottom status bar shows the system clock as 4:06 PM on 3/26/2022.

## Image D

### Discussion:

Similar to the operation executed above using register operands we can see the values used to create the sum that is saved in a new register.

### Third Program:

#### Learning Module 2 Case 2 Step 3 and optional Step 4:

#### Results:

With initial value stored in x10 and 5 initially in memory at location 0x00000078

Expected Final value location:

- Address + Offset = Location
- 0 + 120 = 120 = 0x00000078 hex

The screenshot shows the Venus simulator interface. The main window displays assembly code with columns for PC, Machine Code, Basic Code, and Original Code. The memory dump on the right shows the value 05 at address 0x00000078. The console output is empty.

| PC  | Machine Code | Basic Code     | Original Code   |
|-----|--------------|----------------|-----------------|
| 0x0 | 0x07852483   | lw x9 120(x10) | lw x9, 120(x10) |
| 0x4 | 0x009A84B3   | add x9 x21 x9  | add x9, x21, x9 |
| 0x8 | 0x00148493   | addi x9 x9 1   | addi x9, x9, 1  |
| 0xc | 0x06952C23   | sw x9 120(x10) | sw x9, 120(x10) |

| Address    | +0 | +1 | +2 | +3 |
|------------|----|----|----|----|
| 0x00000094 | 00 | 00 | 00 | 00 |
| 0x00000090 | 00 | 00 | 00 | 00 |
| 0x0000008C | 00 | 00 | 00 | 00 |
| 0x00000088 | 00 | 00 | 00 | 00 |
| 0x00000084 | 00 | 00 | 00 | 00 |
| 0x00000080 | 00 | 00 | 00 | 00 |
| 0x0000007C | 00 | 00 | 00 | 00 |
| 0x00000078 | 05 | 00 | 00 | 00 |
| 0x00000074 | 00 | 00 | 00 | 00 |
| 0x00000070 | 00 | 00 | 00 | 00 |
| 0x0000006C | 00 | 00 | 00 | 00 |

Image E

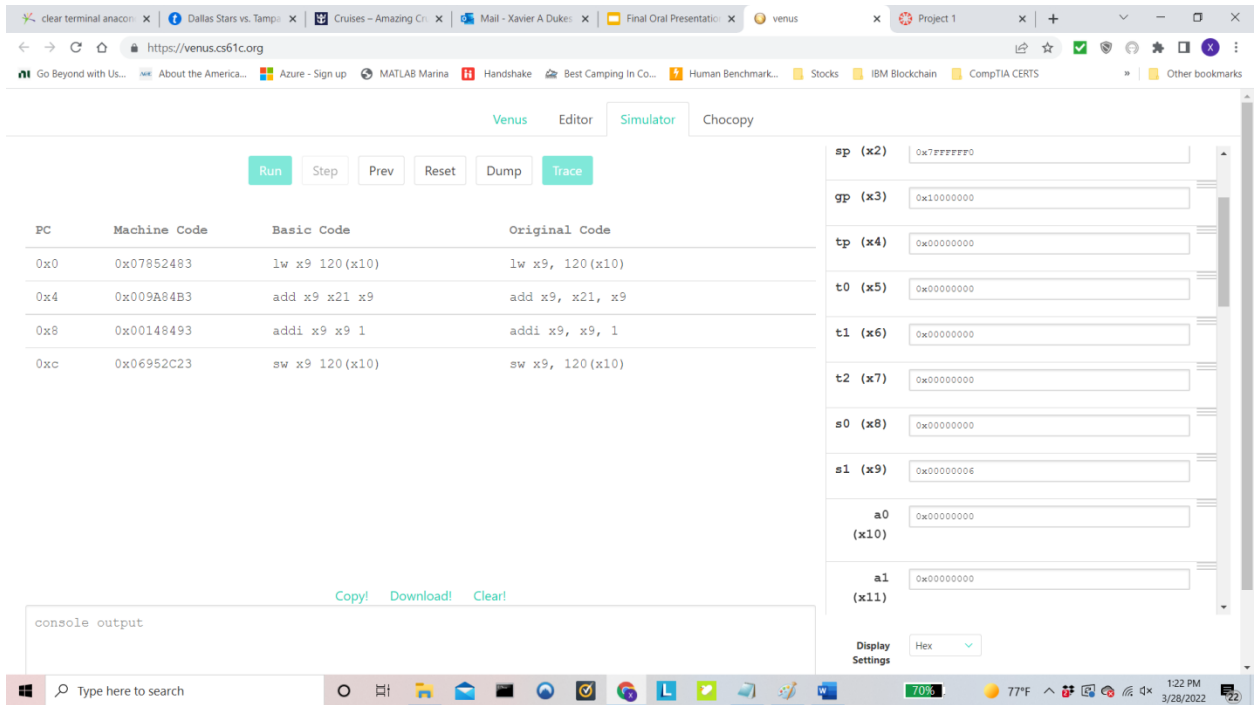


Image F

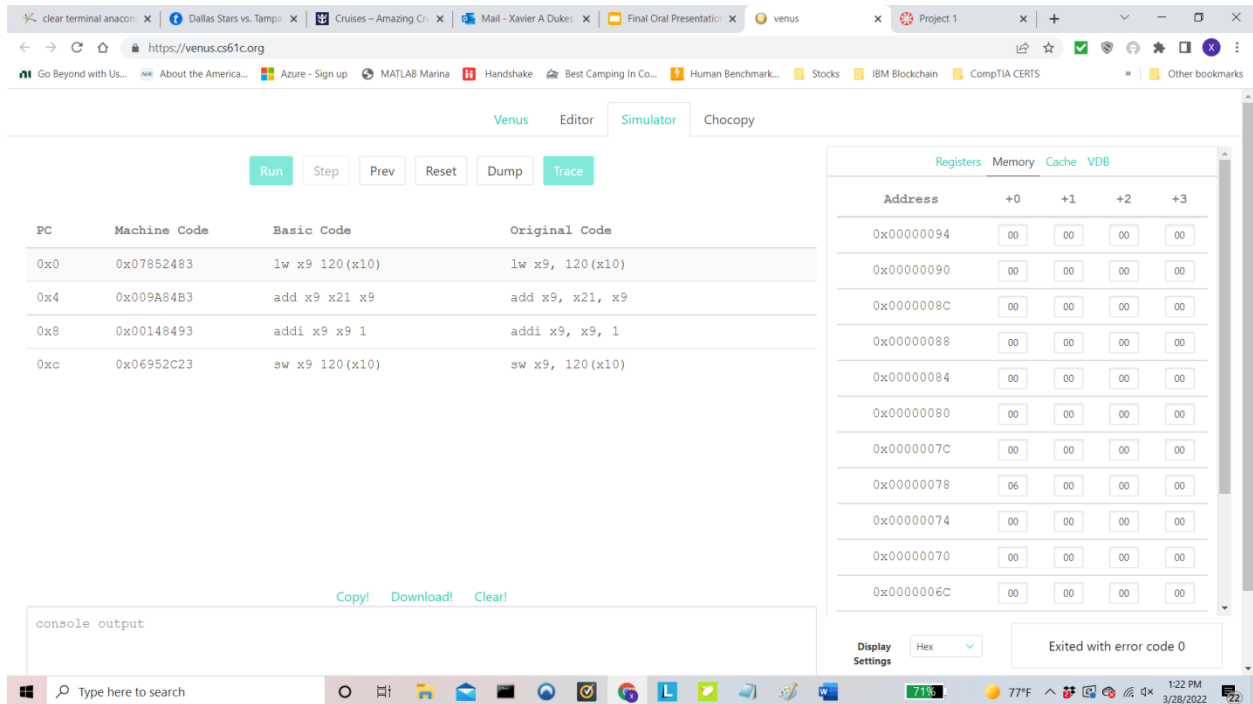


Image G



With initial value 4 stored in x10 and 5 initially in memory at location 0x0000007C

Expected Final value location:

- Address + Offset = Location
- 4 + 120 = 124 = 0x0000007C

The screenshot shows the Venus simulator interface. The main window displays assembly code with columns for PC, Machine Code, Basic Code, and Original Code. The code includes instructions like `lw x9, 120(x10)`, `add x9, x21, x9`, `addi x9, x9, 1`, and `sw x9, 120(x10)`. To the right, the 'Registers' tab is active, showing a memory dump with addresses from 0x00000090 to 0x00000068. The value at address 0x0000007C is 06. The 'Display Settings' dropdown is set to 'Hex'. The bottom status bar shows the system clock as 6:56 PM on 3/28/2022.

| PC  | Machine Code | Basic Code                   | Original Code                |
|-----|--------------|------------------------------|------------------------------|
| 0x0 | 0x07852483   | <code>lw x9, 120(x10)</code> | <code>lw x9, 120(x10)</code> |
| 0x4 | 0x009A84B3   | <code>add x9, x21, x9</code> | <code>add x9, x21, x9</code> |
| 0x8 | 0x00148493   | <code>addi x9, x9, 1</code>  | <code>addi x9, x9, 1</code>  |
| 0xc | 0x06952C23   | <code>sw x9, 120(x10)</code> | <code>sw x9, 120(x10)</code> |

| Address    | +0 | +1 | +2 | +3 |
|------------|----|----|----|----|
| 0x00000090 | 00 | 00 | 00 | 00 |
| 0x0000008C | 00 | 00 | 00 | 00 |
| 0x00000088 | 00 | 00 | 00 | 00 |
| 0x00000084 | 00 | 00 | 00 | 00 |
| 0x00000080 | 00 | 00 | 00 | 00 |
| 0x0000007C | 06 | 00 | 00 | 00 |
| 0x00000078 | 00 | 00 | 00 | 00 |
| 0x00000074 | 00 | 00 | 00 | 00 |
| 0x00000070 | 00 | 00 | 00 | 00 |
| 0x0000006C | 00 | 00 | 00 | 00 |
| 0x00000068 | 00 | 00 | 00 | 00 |

Image H

clear terminal anacon x | Dallas Stars vs. Tamp x | Cruises - Amazing Cr x | Mail - Xavier A Duke x | Final Oral Presentation x | Project 1 x | venus x + - x

https://venus.cs61c.org

Go Beyond with Us... About the America... Azure - Sign up MATLAB Marina Handshake Best Camping In Co... Human Benchmark... Stocks IBM Blockchain CompTIA CERTS Other bookmarks

Venus Editor Simulator Chocopy

Run Step Prev Reset Dump Trace

| PC  | Machine Code | Basic Code     | Original Code   |
|-----|--------------|----------------|-----------------|
| 0x0 | 0x07852483   | lw x9 120(x10) | lw x9, 120(x10) |
| 0x4 | 0x009A84B3   | add x9 x21 x9  | add x9, x21, x9 |
| 0x8 | 0x00148493   | addi x9 x9 1   | addi x9, x9, 1  |
| 0xc | 0x06952C23   | sw x9 120(x10) | sw x9, 120(x10) |

console output

Copy! Download! Clear!

sp (x2) 0x7FFFFFF0

gp (x3) 0x10000000

tp (x4) 0x00000000

t0 (x5) 0x00000000

t1 (x6) 0x00000000

t2 (x7) 0x00000000

s0 (x8) 0x00000000

s1 (x9) 0x00000006

a0 (x10) 0x00000004

a1 (x11) 0x00000000

Display Settings Hex

Type here to search

23% 80°F 6:56 PM 3/28/2022

Image I

With initial value 8 stored in x10 and 8 initially in memory at location 0x00000080

Expected Final value location:

- Address + Offset = Location
- $8 + 120 = 128 = 0x00000080$  hex

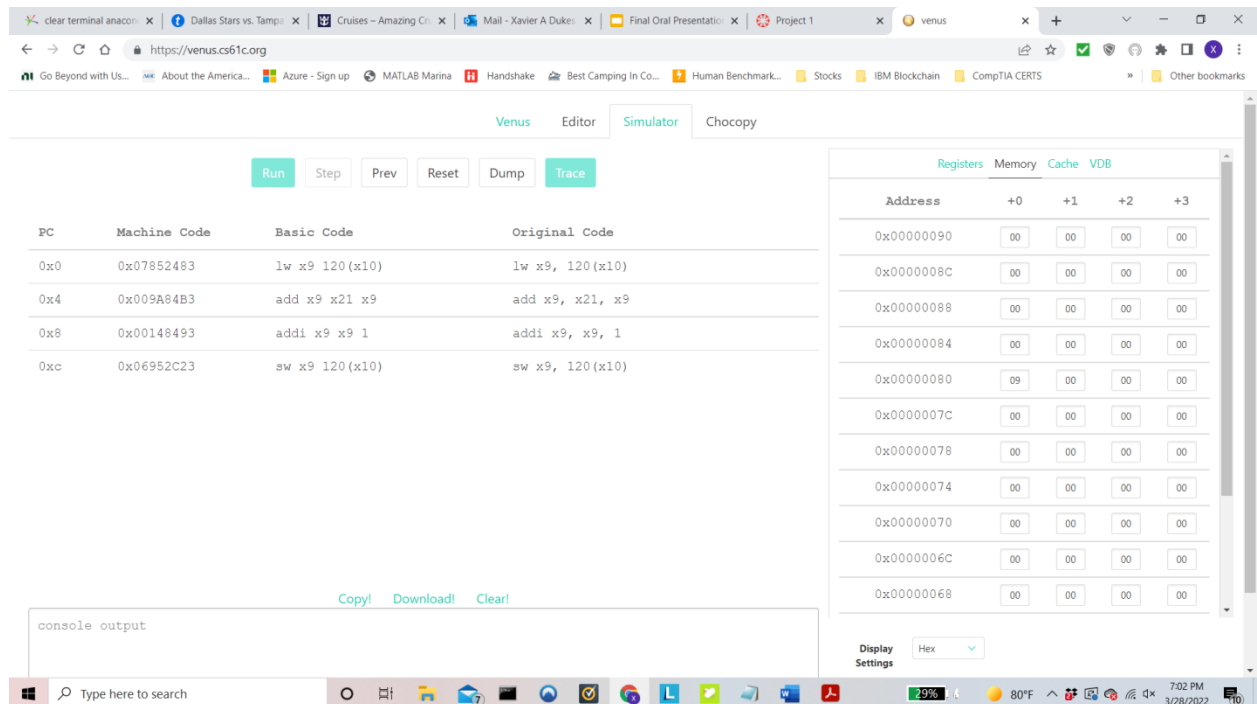
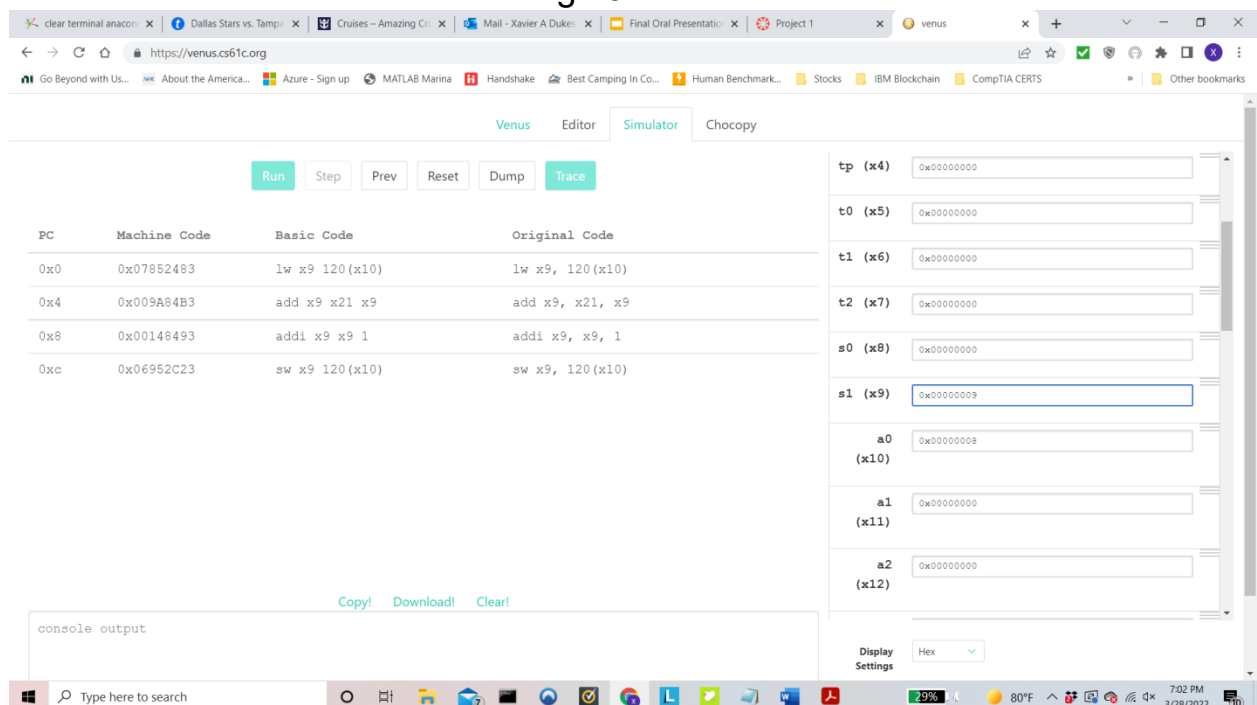


Image J



## Image K

### Discussion:

- The load and store word functions when used store or read a value from a location in memory to a given register or vice versa. These functions use the values in a given register as well as an immediate offset provided to access locations in memory. By changing the initial value in the register, we can alter the address in memory that we access. These exercises allow us to see how memory locations are accessed and computed using offsets.

### Fourth Program:

### Results:

The screenshot displays the Venus simulator interface. The main window shows a table of assembly instructions being executed. The instructions are as follows:

| PC  | Machine Code | Basic Code   | Original Code    |
|-----|--------------|--------------|------------------|
| 0x0 | 0x00100113   | addi x2 x0 1 | addi x2, x0, 1   |
| 0x4 | 0x402080B3   | sub x1 x1 x2 | sub x1, x1, x2   |
| 0x8 | 0x0210A423   | sw x1 40(x1) | sw x1, 40(x1)    |
| 0xc | 0xFE104CE3   | blt x0 x1 -8 | blt x0, x1, loop |

Below the instruction table is a console output area with the text "console output". To the right of the instruction table is a panel for registers, memory, cache, and VDB. The registers panel shows the following values:

| Register | Value      |
|----------|------------|
| zero     | 0x00000000 |
| ra (x1)  | 0x00000000 |
| sp (x2)  | 0x7FFFFFF0 |
| gp (x3)  | 0x10000000 |
| tp (x4)  | 0x00000000 |
| t0 (x5)  | 0x00000000 |
| t1 (x6)  | 0x00000000 |
| t2 (x7)  | 0x00000000 |
| s0 (x8)  | 0x00000000 |

The simulator interface includes tabs for Venus, Editor, Simulator, and Chocopy. The Simulator tab is active, showing the instruction table and register values. The top of the window shows the browser address bar with the URL "https://venus.cs61c.org".

## Image L

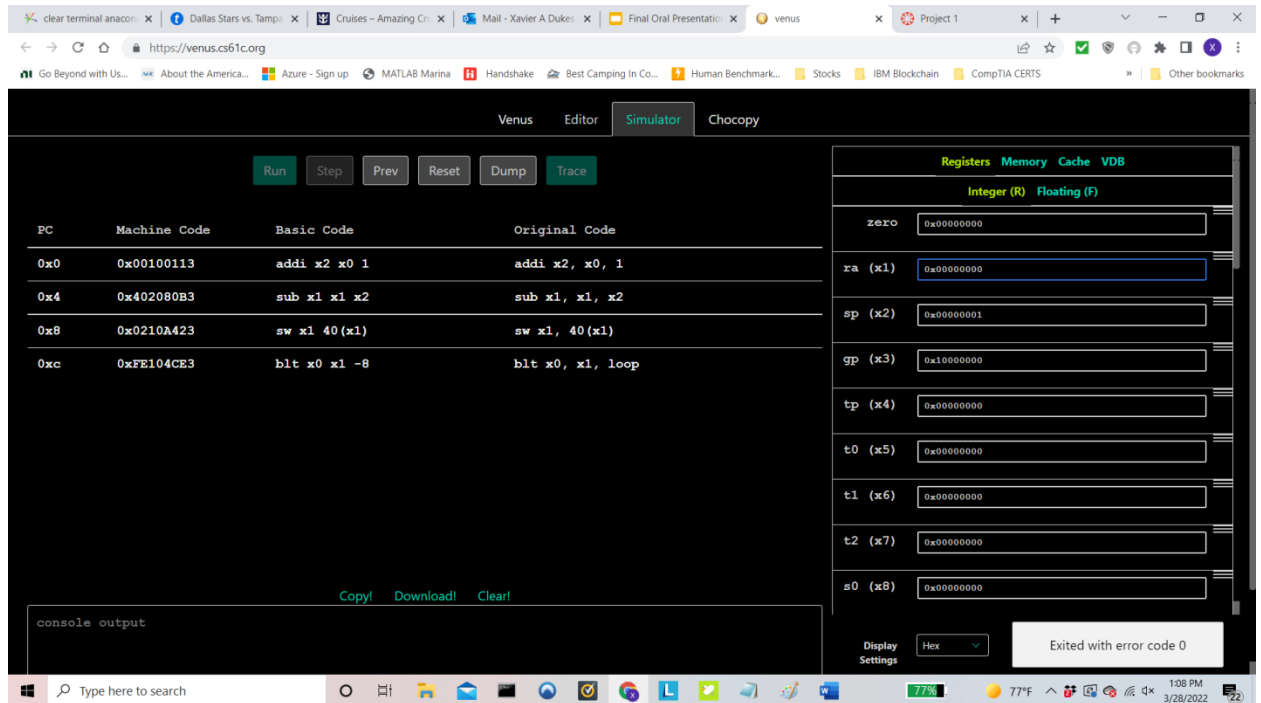


Image M

### Discussion:

This program initializes a value in the first register by adding an immediate value to the x0 register which permanently stores the value of zero. The initial value of reg x1 was initialized prior to this execution. The number in x1 is subtracted by the value in x2 and then stores the value in memory at the location of the value stored in x1 plus a given offset. A conditional Statement then executes until the value in 0x is equal to 1x after several loops of subtraction.

#### 4. Project Question:

Implement  $C = \text{Max}(a, b)$

`addi x9,x9,5 #add 5 to initial register value`

`addi x8,x8,10 #add 10 to initial register value`

`blt x8,x9,max`

`add x10, x0, x8 # if statement above false continue execution assigns max to reg x10`

`blt x0,x8, end # needed to skip over the other assignment of register x9`

`max:`

`add x10, x0, x9`

`end:`

`Before:`

The screenshot displays the Venus MIPS simulator interface. The main window is divided into several sections:

- Machine Code Table:** A table with three columns: Machine Code, Basic Code, and Original Code. It lists six instructions.
- Registers Panel:** A vertical list of registers (zero, ra (x1), sp (x2), gp (x3), tp (x4), t0 (x5), t1 (x6), t2 (x7), s0 (x8), s1 (x9), a0 (x10), a1) with their current values. The s1 (x9) register is highlighted with a green border and contains the value 42.
- Console Output:** A text area at the bottom left labeled "console output" which is currently empty.
- Navigation Buttons:** A row of buttons including Run, Step, Prev, Reset, and Dump.
- Display Settings:** A dropdown menu set to "Decimal".

The Windows taskbar at the bottom shows the system clock as 5:02 PM on 4/3/2022, along with various system icons and a search bar.

| Machine Code | Basic Code                 | Original Code                |
|--------------|----------------------------|------------------------------|
| 0x00548493   | <code>addi x9 x9 5</code>  | <code>addi x9,x9,5</code>    |
| 0x00a40413   | <code>addi x8 x8 10</code> | <code>addi x8,x8,10</code>   |
| 0x00944663   | <code>blt x8 x9 12</code>  | <code>blt x8,x9,max</code>   |
| 0x00800533   | <code>add x10 x0 x8</code> | <code>add x10, x0, x8</code> |
| 0x00804463   | <code>blt x0 x8 8</code>   | <code>blt x0,x8, end</code>  |
| 0x00900533   | <code>add x10 x0 x9</code> | <code>add x10, x0, x9</code> |

Registers Memory

| Register | Value      |
|----------|------------|
| zero     | 0          |
| ra (x1)  | 0          |
| sp (x2)  | 2147483632 |
| gp (x3)  | 268435456  |
| tp (x4)  | 0          |
| t0 (x5)  | 0          |
| t1 (x6)  | 0          |
| t2 (x7)  | 0          |
| s0 (x8)  | 2          |
| s1 (x9)  | 42         |
| a0 (x10) | 0          |
| a1       | 0          |

Display Settings: Decimal

After:

The screenshot shows the Venus MIPS simulator interface. The main window is divided into several sections:

- Editor/Assembler:** Contains a table with three columns: Machine Code, Basic Code, and Original Code.
- Registers:** A panel on the right showing the current values of MIPS registers.
- Console Output:** A text area at the bottom left for displaying program output.

| Machine Code | Basic Code    | Original Code   |
|--------------|---------------|-----------------|
| 0x00548493   | addi x9 x9 5  | addi x9,x9,5    |
| 0x00a40413   | addi x8 x8 10 | addi x8,x8,10   |
| 0x00944663   | blt x8 x9 12  | blt x8,x9,max   |
| 0x00800533   | add x10 x0 x8 | add x10, x0, x8 |
| 0x00804463   | blt x0 x8 8   | blt x0,x8, end  |
| 0x00900533   | add x10 x0 x9 | add x10, x0, x9 |

| Register | Value      |
|----------|------------|
| zero     | 0          |
| ra (x1)  | 0          |
| sp (x2)  | 2147483632 |
| gp (x3)  | 268435456  |
| tp (x4)  | 0          |
| t0 (x5)  | 0          |
| t1 (x6)  | 0          |
| t2 (x7)  | 0          |
| s0 (x8)  | 12         |
| s1 (x9)  | 47         |
| a0 (x10) | 47         |
| a1       | 0          |

Discussion: The code above takes two values from registers x8 and x9 and compares them with the blt function. If this statement is found to be true the code branches and assigns the x9 register to x10 as it was shown to be greater than register x8. If this statement is false the code will assign x8 to register x10. Since it is now known that x8 is greater than x9 we use this knowledge to create another branch statement to skip over the section labeled max.



## Design Square Sum calculator:

```
sumsquare:      addi sp, sp, -12 #make space for 3 words
```

```
                sw ra, 0(sp)
```

```
                sw s0, 4(sp) # state registers s0
```

```
                sw s1, 8(sp) # state registers s1
```

```
                add s0,a0,x0 # set equal to value n
```

```
                add s1,x0,x0 # set s1 equal to 0
```

```
loop:
```

```
                bge x0, s0, end
```

```
                add a0,s0,x0
```

```
                jal ra, sumsquare #call square function
```

```
                add s1,s1,a0
```

```
                addi s0,s0,-1
```

```
                jal x0, loop #jump back to loop
```

```
end:            add a0,s1,x0
```

```
                lw ra, 0(sp)
```

```
                lw s0,4(sp)
```

```
                lw s1,8(sp)
```

Before:

The screenshot displays the Venus MIPS simulator interface. The top section shows the browser tabs and the URL <https://venus.kvakil.me>. The main interface is divided into two tabs: "Editor" and "Simulator". The "Simulator" tab is active, showing a table of assembly code with columns for Machine Code, Basic Code, and Original Code. The code is as follows:

| Machine Code | Basic Code     | Original Code                                       |
|--------------|----------------|-----------------------------------------------------|
| 0xff410113   | addi x2 x2 -12 | sumsquare: addi sp, sp, -12 #make space for 3 words |
| 0x00112023   | sw x1 0(x2)    | sw ra, 0(sp)                                        |
| 0x00812223   | sw x8 4(x2)    | sw s0, 4(sp) # state registers s0                   |
| 0x00912423   | sw x9 8(x2)    | sw s1, 8(sp) # state registers s1                   |
| 0x00050433   | add x8 x10 x0  | add s0,a0,x0 # set equal to value n                 |
| 0x000004b3   | add x9 x0 x0   | add s1,x0,x0 # set s1 equal to 0                    |
| 0x00805c63   | bge x0 x8 24   | bge x0, s0, end                                     |
| 0x00040533   | add x10 x8 x0  | add a0,s0,x0                                        |
| 0xfelf00ef   | jal x1 -32     | jal ra, sumsquare #call square function             |

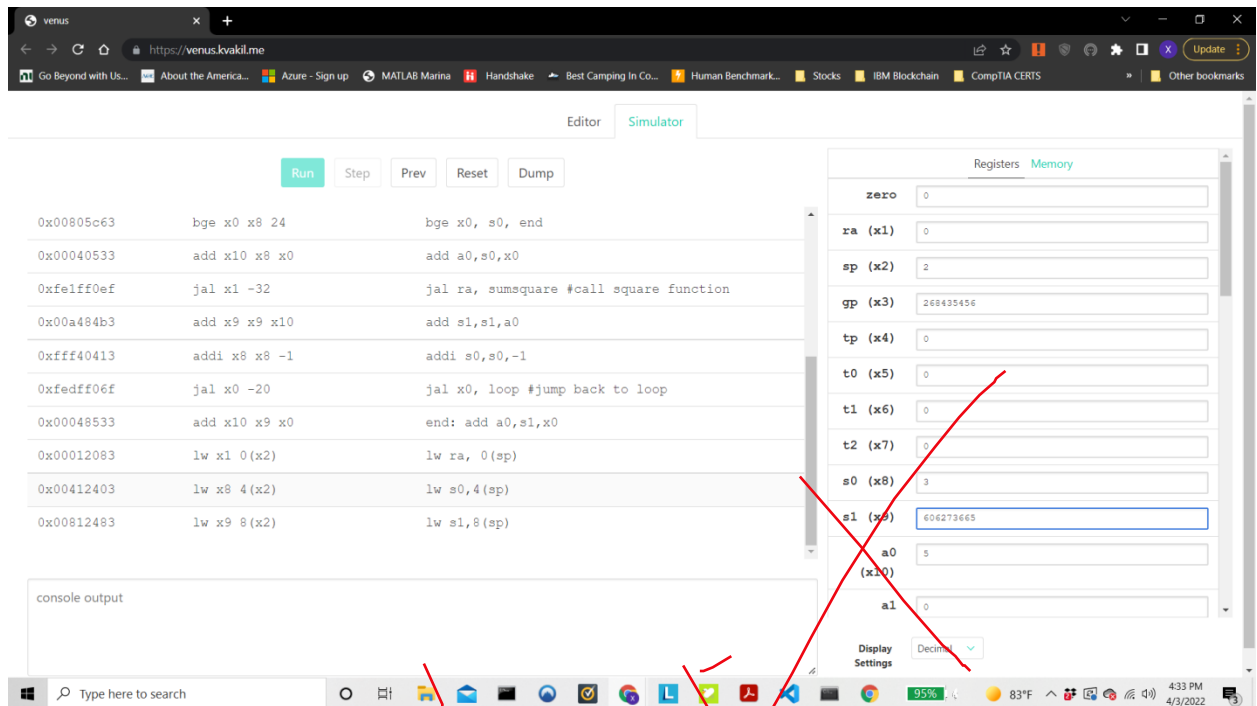
Below the code table is a "console output" section. On the right side, there is a "Registers" panel showing the values of various registers:

| Register | Value     |
|----------|-----------|
| zero     | 0         |
| ra (x1)  | 0         |
| sp (x2)  | 2         |
| gp (x3)  | 268435456 |
| tp (x4)  | 0         |
| t0 (x5)  | 0         |
| t1 (x6)  | 0         |
| t2 (x7)  | 0         |
| s0 (x8)  | 40        |
| s1 (x9)  | 30        |
| a0 (x10) | 5         |
| a1       | 0         |

The "Display Settings" section at the bottom right shows "Decimal" as the selected format. The bottom of the screen shows the Windows taskbar with the search bar and various application icons.



After:



Discussion:

First the registers are stored to memory and space is allocated for the values. Inside the loop, values are continuously added to represent multiplication based off the value being multiplied and then, the value in S0 is decremented by one and the value n-1 is repeated added to simulate multiplication and this continues until s0 is at 0 at which point the end is reached at which point it is all stored in register a0 and the initial values in register SP are reinitialized.

5. Conclusion:

The RISC-V architecture uses simple instructions and makes use of various methods to access values in registers and in memory. These can be used in a variety of ways to execute a countless array of unique operations. Students learned about the use of registers and the accessing of various memory locations, offsets, conditional statements, and their relation to assembly language. Students gained an increase knowledge of the Venus simulation software and the format associated with machine code instructions of various types.

**BELOW AVERAGE**

**Below Average Student Original Graded Assignment**

**Prairie View A&M University**  
**Electrical Engineering Department**



6.5

**Juwan Byers**  
**ELEG 4339 Computer Organization and Design**  
**Dr. Li and Dr. Cui**  
**Project 1**  
**4/5/2022**

## Goals and Procedure

The goals of this project is to get familiar with Venus simulator, observe how registers and memory support computing, and review RISC-V assembly programming.

### Module 1

- Type "add x9, x20, x21" in Editor field.
- Put 1 as initial value of x20 and 2 as initial value of x21.
- Click "Run" to observe result in x9.
- Click "Reset" to see all registers go back to zeros.
- Switch to "Memory". Observe the machine code stored at the beginning of the memory.
  - o 0x00000000 b3 04 5a 01
- Click "Dump". The machine code is copied in clipboard and shown in output.
- What is the endian choice of RISC-V? Verify with the binary code as shown in textbook page 88.

RunStepPrevResetDumpTrace

| PC  | Machine Code | Basic Code     | Original Code  |
|-----|--------------|----------------|----------------|
| 0x0 | 0x015A04B3   | add x9 x20 x21 | add x9,x20,x21 |

s1  
(x9)

0x00000011

s4  
(x20)

0x00000001

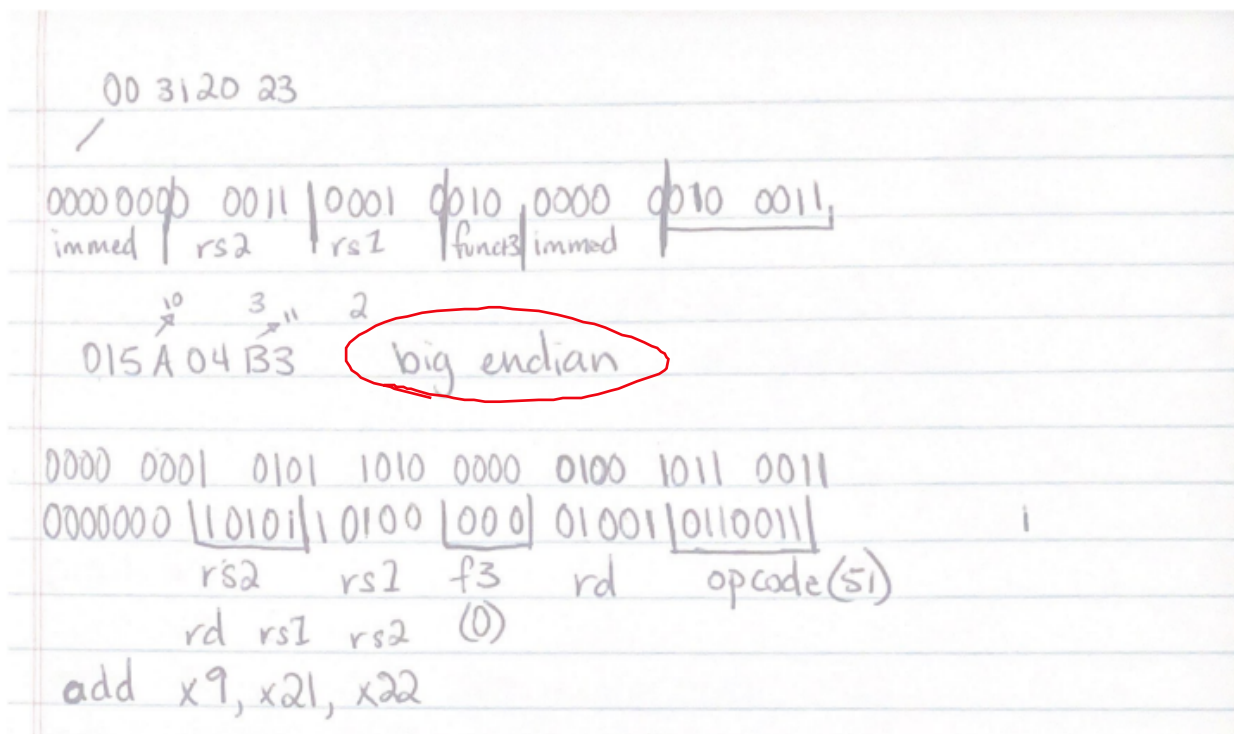
s5  
(x21)

0x00000010

| Registers  | Memory | Cache | VDB |    |
|------------|--------|-------|-----|----|
| Address    | +0     | +1    | +2  | +3 |
| 0x00000018 | 00     | 00    | 00  | 00 |
| 0x00000014 | 00     | 00    | 00  | 00 |
| 0x00000010 | 00     | 00    | 00  | 00 |
| 0x0000000C | 00     | 00    | 00  | 00 |
| 0x00000008 | 00     | 00    | 00  | 00 |
| 0x00000004 | 00     | 00    | 00  | 00 |
| 0x00000000 | B3     | 04    | 5A  | 01 |

Copy! Download! Clear!

0x015A04B3



## Module 2 Case 1

- Give initial values to registers: x1 = 3, x2 = 4.
- Type "ADD x3, x2, x1" in Editor.

- Click "Run". Get a screenshot from the interface to show the result in x3.
- Work out machine code based on what you learned from Chapter 2. Analyze and verify the machine code obtained from the simulator

|            |             |              |              |    |    |    |
|------------|-------------|--------------|--------------|----|----|----|
| 0x4        | 0x001101B3  | add x3 x2 x1 | add x3,x2,x1 |    |    |    |
|            |             | Address      | +0           | +1 | +2 | +3 |
|            |             | 0x00000018   | 00           | 00 | 00 | 00 |
|            |             | 0x00000014   | 00           | 00 | 00 | 00 |
|            |             | 0x00000010   | 00           | 00 | 00 | 00 |
|            |             | 0x0000000C   | 00           | 00 | 00 | 00 |
|            |             | 0x00000008   | 00           | 00 | 00 | 00 |
|            |             | 0x00000004   | B3           | 01 | 11 | 00 |
| ra<br>(x1) | 0x00000011  |              |              |    |    |    |
| sp<br>(x2) | 0x000000100 |              |              |    |    |    |
| gp<br>(x3) | 0x000000111 |              |              |    |    |    |

## Module 2 Case 2

- Give initial value x10 = 4.
- Type following code to "Editor".
  - o lw x9, 120(x10)
  - o add x9, x21, x9
  - o addi x9, x9, 1
  - o sw x9, 120(x10)
- Simulate the code using single steps. Observe the changes to registers. At last, verify the final memory location value and show the calculation of memory address as the source and destination of lw and sw

| PC  | Machine Code | Basic Code     | Original Code   |
|-----|--------------|----------------|-----------------|
| 0x0 | 0x07852483   | lw x9 120(x10) | lw x9, 120(x10) |
| 0x4 | 0x009A84B3   | add x9 x21 x9  | add x9,x21,x9   |
| 0x8 | 0x00148493   | addi x9 x9 1   | addi x9,x9,1    |
| 0xc | 0x06952C23   | sw x9 120(x10) | sw x9,120(x10)  |

| PC  | Machine Code | Basic Code     | Original Code   |
|-----|--------------|----------------|-----------------|
| 0x0 | 0x07852483   | lw x9 120(x10) | lw x9, 120(x10) |
| 0x4 | 0x009A84B3   | add x9 x21 x9  | add x9,x21,x9   |
| 0x8 | 0x00148493   | addi x9 x9 1   | addi x9,x9,1    |
| 0xc | 0x06952C23   | sw x9 120(x10) | sw x9,120(x10)  |

— 05

| Address    | +0 | +1 | +2 | +3 |
|------------|----|----|----|----|
| 0x0000001C | 00 | 00 | 00 | 00 |
| 0x00000018 | 00 | 00 | 00 | 00 |
| 0x00000014 | 00 | 00 | 00 | 00 |
| 0x00000010 | 00 | 00 | 00 | 00 |
| 0x0000000C | 23 | 2C | 95 | 06 |
| 0x00000008 | 93 | 84 | 14 | 00 |
| 0x00000004 | B3 | 84 | 9A | 00 |
| 0x00000000 | 83 | 24 | 85 | 07 |

s1  
(x9)

0x00000001

a0  
(x10)

0x00000004



001101B3

0000 0000 0001 0001 0000 0001 1011 0011

00000000 | 000011 | 00010 | 000 | 00011 | 01100111

rs2 rs1 f3 rd opcode

rd rs1 rs2 (0) (51)

add x3, x2, x1

06952C23

0000 0110 1001 0101 0010 1100 0010 0011

0000011 01001 01010 010 1000 0100011

immed[11:5] rs2 rs1 f3 immed opcode (35)

3 9 10 0 24

sw x9, 120(x10)

00148493

0000 0000 0001 0100 1000 0100 1001 0011

000000000001 01001 000 0100 0010011

immediate rs1 f3 rd opcode

1 9 0 9 19

addi x9, x9, 1

009A84B3

00000000 1001 1010 1000 0100 1011 0011

0000000 01001 10101 000 01001 0110011

f7 rs2 rs1 f3 rd opcode

0 9 21 0 9 51

add x9, x21, x9

0785 2483

0101

0000 0111 1000 ~~1001~~ 0010 0100 1000 0011

[000001111000] [01010] [010] [01001] [0000011]

immed

rs1

fs

rd

op

120

(10)

(2)

(9)

(3)

#### Part 4

s1

0x00000001

(x9)

a0

0x00000002

(x10)

| Address    | +0 | +1 | +2 | +3 |
|------------|----|----|----|----|
| 0x00000028 | 00 | 00 | 00 | 00 |
| 0x00000024 | 00 | 00 | 00 | 00 |
| 0x00000020 | 00 | 00 | 00 | 00 |
| 0x0000001C | 00 | 00 | 00 | 00 |
| 0x00000018 | 00 | 00 | 00 | 00 |
| 0x00000014 | 00 | 00 | 00 | 00 |
| 0x00000010 | 00 | 00 | 00 | 00 |
| 0x0000000C | 23 | 2C | 95 | 06 |
| 0x00000008 | 93 | 84 | 14 | 00 |
| 0x00000004 | B3 | 84 | 9A | 00 |
| 0x00000000 | 83 | 24 | 85 | 07 |

**s1**   
**(x9)**

**a0**   
**(x10)**

**s1**   
**(x9)**

**a0**   
**(x10)**

Exercise 3

ra 0x0000000E  
(x1)

---

sp 0x00000001  
(x2)

After 1<sup>st</sup> step

ra 0x0000000D  
(x1)

---

sp 0x00000001  
(x2)

After 2<sup>nd</sup> step

ra 0x0000000D  
(x1)

---

sp 0x00000001  
(x2)

```

#include <stdio.h>
int main () {
    int a, b, c, 1;

    c = a + 1;
    b = b - c;
    if (a < b) {
        // 
    }
}

```

X

Main Question

Part 1

```

1 blt x1,x2,Else
2 sw x3,0(x1)
3 Else:sw x3,0(x2)
4 Exit:

```

flow chart ?

- 05

| PC  | Machine Code | Basic Code          | Original Code       |
|-----|--------------|---------------------|---------------------|
| 0x0 | 0x00000063   | beq x0 x0 0         | beq x0 x0 0         |
| 0x4 | 0x0030A000   | Invalid Instruction | Invalid Instruction |
| 0x8 | 0x00312023   | sw x3 0(x2)         | Else:sw x3,0(x2)    |

ra  
(x1) 0x00000000

sp  
(x2) 0x00000001

gp  
(x3) 0x00000000

Else statement true

| PC  | Machine Code | Basic Code  | Original Code    |
|-----|--------------|-------------|------------------|
| 0x0 | 0x0020C463   | blt x1 x2 8 | blt x1,x2,Else   |
| 0x4 | 0x0030A023   | sw x3 0(x1) | sw x3,0(x1)      |
| 0x8 | 0x00312023   | sw x3 0(x2) | Else:sw x3,0(x2) |

ra  
(x1) 0x00000000

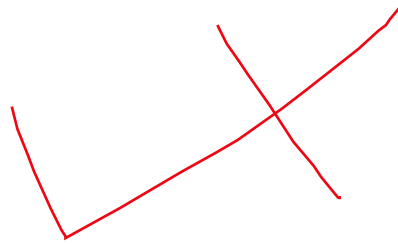
sp  
(x2) 0x7FFFFFF0

gp  
(x3) 0x10000000

Part 2

```
1 addi sp,sp,-12
2 sw ra,0(sp)
3 sw s0,4(sp)
4 sw s1,8(sp)
5 add s0,a0,x0
6 add s1,x0,x0
7 loop:bge x0,s0,end
8 add a0,s0,x0
9 jal ra,square
10 add s1,s1,a0
11 addi s0,s0,-1
12 jal x0,loop
13 end: add a0,s1,x0
14 lw ra,0(sp)
15 lw s0,4(sp)
16 lw s1,8(sp)
17 addi sp,sp,12
18 jr ra
```

-1.5





# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**SPRING 2022**

**Title of Assignment: Final Exam**

## **TOPICS COVERED IN FINAL EXAM**

**Final exam is a comprehensive exam.  
It covered the following topics:**

1. Computer abstractions and technology
2. Computer Performance; define computer performance in terms of CPU time; CPI
3. Instructions: Language of the computer; RISC-V assembly language
4. Arithmetic for computer
5. The processor
6. Large and Fast: Exploiting Memory Hierarchy

# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**Final Exam**

**ABOVE AVERAGE**

**Above Average Student Original Graded Assignment**

**ELEG 4339 Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Final Exam

Spring 2022

Name: (Print) John Barrett

(Sign) John W. Barrett

|              |     |      |
|--------------|-----|------|
| 1            | 16  | 16   |
| 2            | 16  | 12.5 |
| 3            | 12  | 12   |
| 4            | 20  | 18   |
| 5            | 18  | 17.5 |
| 6            | 10  | 9    |
| 7            | 8   | 4    |
| <b>Total</b> | 100 | 89   |

16  
2

(16pts) 1. Short answer (2 points each)

(2 pts) (1.1) Of all the available datapath components, please specify (place a checkmark) whether the component will be needed for the execution of instruction: `addi x10, x12, 1`

Yes NO

- |                                     |                                     |                           |
|-------------------------------------|-------------------------------------|---------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Instruction memory        |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | ALU                       |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Register file             |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Immediate generation unit |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | Data memory               |

(2 pts) (1.2) For each of the following statement, place a checkmark in the column identifying which memory technology, SRAM or DRAM, the statement best describes.

SRAM DRAM

- |                                     |                                     |                                            |
|-------------------------------------|-------------------------------------|--------------------------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | is used for cache memory                   |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | is used for main memory                    |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | is the faster one of the two technologies  |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | is the cheaper one of the two technologies |

(2 pts) (1.3). Computer architect divide processors based on design criteria of instruction sets into two broad categories: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Choose **all** the following characteristics that **are applicable to RISC processor**.

- a. Variable-length instructions
- ☒ b. Fixed-length instructions
- c. Many instructions (typically, hundreds)
- ☒ d. Strives for a minimum instruction set
- ☒ e. Each instruction performs a basic computation
- f. Each instruction can perform an arbitrarily complex computation

(2 pts) (1.4) Please describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

*A compiler is used to turn "C" into assembly then an assembler and a linker are used to turn assembly into working machine code.*

(2 pts) (1.5). Assume that one processor has **13 pipeline stages**, assume it is perfectly pipelined and the program is very long, how much speed up will it achieve compared to a nonpipelined processor.

Answer 13 times

(2 pts) (1.6). Derived from the paper-and-pencil method learned in grammar school, integer multiplication in computer is accomplished by iterative adds and shifts.

(2 pts) (1.7). The single-cycle datapath conceptually described in the class and in the textbook must have separate **Instruction and Data memories** because (single choice)

- a. The format of data and instructions is different in RISC-V and hence different memory are needed.
- b. having separate memories is less expensive.
- c. The processor operate in one cycle and cannot use a single-ported memory for two different accesses within that cycle.

(2 pts) (1.8). Name the **special purpose register** that stores the **address** of the next instruction to be fetched from memory.

Answer: Program Counter

## (16pts) 2. Concepts Understanding (4 points each)

(2.1) There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards. Please name what are the **three different types of hazards**?

Data Hazard  
Structure Hazard  
the access Hazard  
store/use

(2.2). It is the principle of locality that gives us a chance to overcome the long latency of memory access. Please explain what are **temporal locality** and **spacial locality**, and give an very simple example (in the area of computer architecture) to illustrate the point.

Temporal Locality - is when the instructions are used close in time to the reference. Example: when doing lots of math loading all data/variables that will be used now and close in future is good.

Spacial Locality - when the data is physically close to reference in memory.

(2.3) When determine the CPU performance, CPU time is related to Instruction count, CPI and clock cycle. Please fill out the following table to check which aspects are affected by the specific hardware and software component.

|                      | Instruction count | CPI | Clock cycle |
|----------------------|-------------------|-----|-------------|
| Algorithm            | ✓ ✓               | ✓   | ✓ X         |
| ISA                  | 0                 | ✓   | ✓ ✓         |
| Compiler             | ✓ ✓               | ✓   | ✓ X         |
| Programming language | ✓ ✓               | ✓   | ✓ X         |

Example: when dealing with arrays loading the whole array when is stored sequentially is good.

(2.4) Could you briefly explain/demonstrate Memory Hierarchy in a typical PC setting.

Cache is small and fast, used to store data immediately being used by the CPU.

Main memory is still fast not as fast as cache but is used to store data that is being used by the program but not immediately so. Bulk storage is slow but large used to store data long-term.

(12 pts) 3 Assuming an Instruction cache miss rate for a program is 1%, and a Data cache miss rate is 5%. If a processor has a ideal CPI of 2 without any memory stalls, and the miss penalty is 100 cycles for all misses. Assume that: 30% of the instructions are load & stores, 50% of the instructions are ALUs, and 20% of the instructions are controls.

(6 pts) 3.1

- (i) What is actual CPI with the possible cache misses?  $2 + (0.01)(100) + (0.05)(0.3)(100) = 4.5$
- (ii) Assume that we have perfect data and instruction cache (no cache misses), what is the performance speed up of the CPU with the ideal perfect cache vs. the CPU with actual cache.

$$\frac{4.5}{2} = 2.25 \text{ times}$$

(6 pts) 3.2 What if the processor is faster, but memory system is not. Suppose we speed up the ideal CPI without any memory stalls from 2 to 1 without changing the clock rate.

- (i) What is the CPI of the system with cache misses?  $1 + 1 + 1.5 = 3.5$
- (ii) What is the performance speed up of the CPU with the ideal perfect cache vs. the CPU with actual cache?  $\frac{3.5}{1} = 3.5 \text{ times}$
- (iii) What is your observation on the effect of the memory on the system performance?

That the faster you make the CPU the more bottlenecked the performance if the memory system is not improved.



18  
2

(20pts) 4.

(4 pts) (4.1) Given a 32-bit binary number, please demonstrate the range it could represent if it represent a single-precision IEEE 754 floating-point number.

$$1 + \text{Frac} \times 2^{(\text{Exp} - 127)}$$

$$[-2^{126}, +2^{127}]$$

if Exp is small = 1 and Frac = 0

if Exp is big = 254 and Frac = .111111 then  $1 + \text{Frac} \approx$

4

(4 pts) (4.2). What is overflow and underflow for floating point arithmetic?

overflow is when the absolute value of the number is greater than the limits of the range.

underflow is when the absolute value of the number is too small to be represented.

6

(6 pts) (4.3) What is the decimal value of this IEEE 754 single-precision floating point number

$$\begin{array}{l} \boxed{0} \boxed{11000} \boxed{1010111011000000000000} \\ \boxed{0} \boxed{1110001} \boxed{1010111011000000000000} \\ + 2^{27} \end{array}$$

$$= (1.010111011) \times 2^{227-127} = 1.2805 E 30$$

6

(6 pts) (4.4) Please represent the decimal number: -188.875 using single-precision floating point IEEE 754 standard.

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

$$S = 1$$

$$188 = 10111100$$

$$0.875 = 0.75 + 0.125$$

$$0.75 = 0.5 + 0.25$$

$$0.875 = 0.5 + 0.25 + 0.125$$

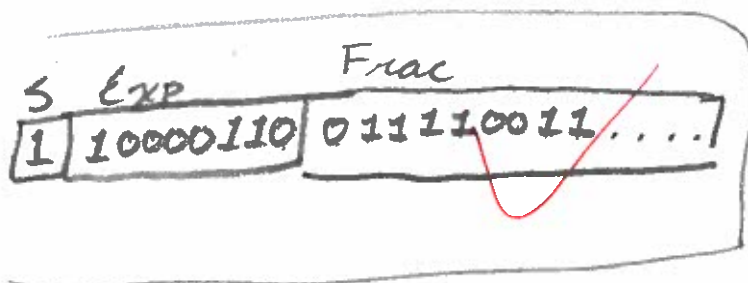
$$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3}$$

$$188.875 = 10111100.111$$

$$\text{Frac} = 0111100111$$

$$7 = \text{Exp} - 127 \quad \text{Exp} = 134$$

$$134 = 10000110$$



**(18 pts) 5. Single cycle implementation – multi cycle implementation – pipelined implementation:**

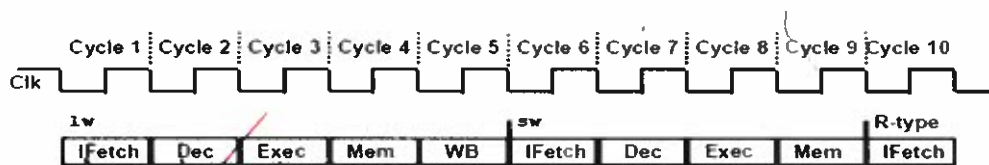
Calculate cycle time assuming negligible delays (for muxes, control unit, immediate Gen, PC access, shift left 1, wires) except: Instruction and Data Memory (200ps); ALU and adders (200ps); and Register File access (reads or writes) (100 ps)

- (5 pts) (5.1) (i) Please fill in the table and calculate the total time needed for each instruction  
(ii) What will be a good time for a single cycle time (2pts)?

| Instruction Class    | Instruction memory | Register Read | ALU Operation | Data Mem | Register Write | Total |
|----------------------|--------------------|---------------|---------------|----------|----------------|-------|
| R type( such as ADD) | 200                | 100           | 200           | —        | 100            | 600   |
| Load (lw)            | 200                | 100           | 200           | 200      | 100            | 800   |
| Store (sw)           | 200                | 100           | 200           | 200      | —              | 700   |
| BEQ                  | 200                | 100           | 200           | —        | —              | 500   |

400 ps

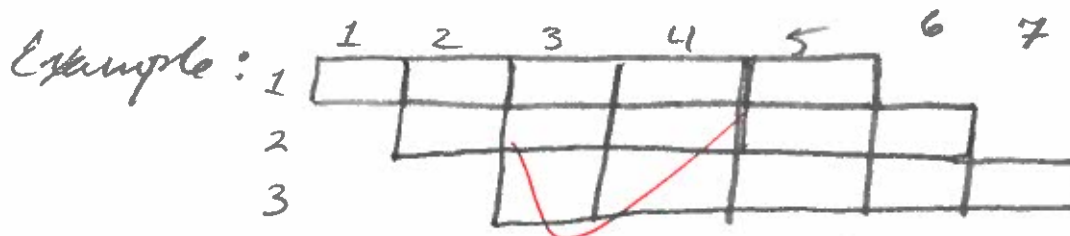
- (2 pts) (5.2) when we adopt a multicycle implementation. Each step will take 1 clock cycle, please explain what will be an appropriate cycle time?

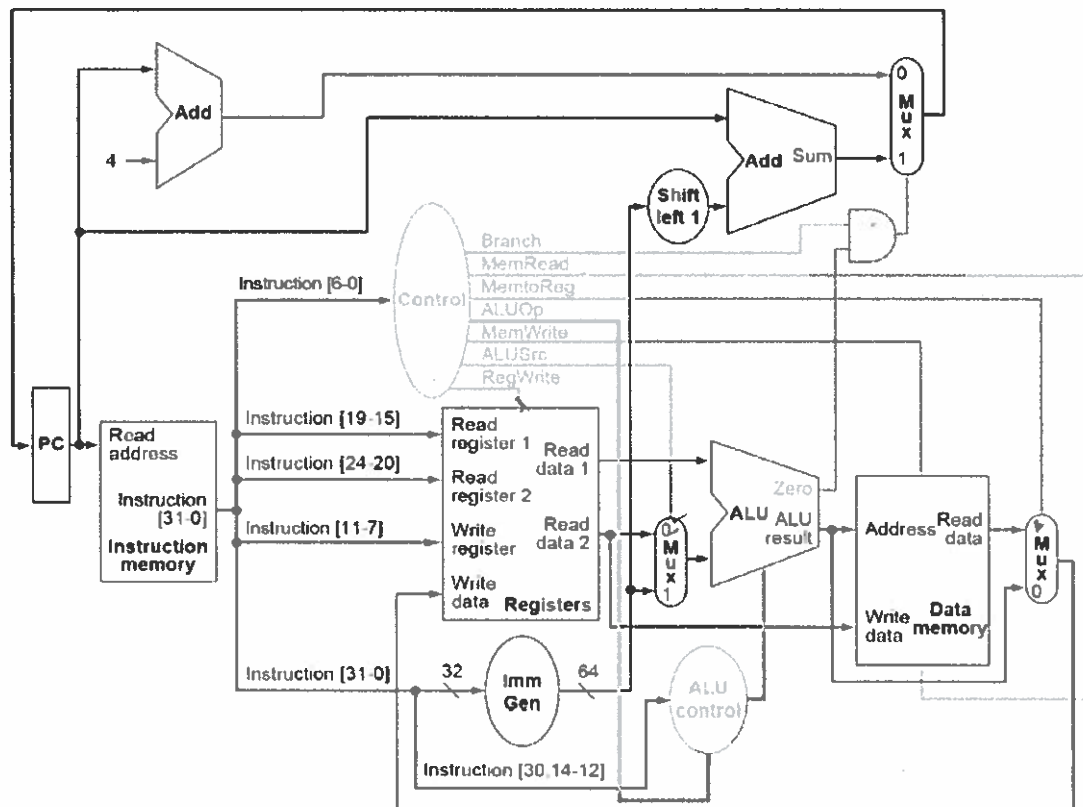


200ps because that is the longest any step requires

- (3 pts) (5.3) Please explain what is the pipelined implementation.

Pipe line implementation allows for hardware resources to be used on the next instruction as soon as it is available (once the previous instruction is done with it)





4 (4 pts) 5.4. Please fill in the following table, which specify how to set the control signals when executing the 2 instructions.

| Instruction | Branch | MemtoReg | ALUSrc | RegWrite | MemRead | MemWrite | ALUOp |
|-------------|--------|----------|--------|----------|---------|----------|-------|
| ld          | 0      | 1        | 1      | 1        | 1       | 0        | 00    |
| Beq         | 1      | 0X       | 0      | 0        | 0       | 0        | 01    |

3.5 (4 pts) 5.5. Please explain the detailed execution steps for the instruction:

sd x10, 40(x11)

Fetch instruction; Decode; Read cache x10 & x11;  
Create immediate out of "40" use ALU to create  
Data memory address from x11 and "40", Write x10's  
data to memory address.

9 (10 pts) 6. The following RISC-V code segment is corresponding to the C code:  
 $A = B - E; \quad C = B + F;$

1. ld x1, 0(x6)  
 2. ld x2, 8(x6)  
 3. sub x3, x1, x2  
 4. sd x3, 16(x6)  
 5. ld x4, 24(x6)  
 6. add x5, x1, x4  
 7. sd x5, 40(x6)

5 stage assumed  
 $4 + 7 + 2 = 13$

4 (4 pts) 6.1 Assume forwarding (aka bypassing) scheme is adopted here. (i) Please specify the data hazards even with the forwarding scheme and point out where stalls should be added. (ii) how many clock cycles do you need to implement the above code sequence?

i) H1 & H3 exist even w/ forwarding. stalls should be added between I2 & I3 as well as between I5 & I6.

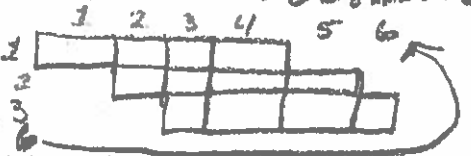
ii) 13 cycles (assuming 5 stage)

3 (4pts) 6.2 (i) Please reorder the code segment (while do not change the function of the program) to avoid the stalls. (ii) how many clock cycles do you need to implement the above code sequence?

ld x1 0(x6)    sd x3 16(x6)  
 ld x2 8(x6)    add x5 x1 x4  
 ld x4 24(x6)    sd x5 40(x6)  
 sub x3 x1 x2

2 (2pts) 6.3 What is the minimum number of cycles needed to completely execute N instructions on a CPU with a K stage pipeline? Justify your formula.

$(K-1) + N = \text{# of Cycles}$ . Justification: minimum = no stalls  
 $K=4$      $N=3$



4 (8 pts) 7. Please change the following c statement to RISC-V statements.

Assume that base address of array A is in register x10, base address of array B is in register x11, and each array element is a doubleword (64 bit). Variables register allocation are:

c: x6, i: x7, and j: x8.

What is the RISC-V assembly code for the C statement?

$c = A[i] - B[j]$

addi x9, x0, 8  
 mul x12, x7, x9  
 mul x13, x8, x9  
 add x14, x12, x10  
 add x15, x13, x11  
 ld x16, 0(x14)  
 ld x17, 0(x15)  
 sub x6, x16, x17

**AVERAGE**

## **Average Student Original Graded Assignment**

**ELEG 4339 Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Final Exam

Spring 2022

Name: (Print) John Owens

(Sign) John Owens

|              |     |      |
|--------------|-----|------|
| 1            | 16  | 15.5 |
| 2            | 16  | 15.5 |
| 3            | 12  | 6.5  |
| 4            | 20  | 15   |
| 5            | 18  | 15   |
| 6            | 10  | 8.5  |
| 7            | 8   | 6    |
| <b>Total</b> | 100 | 82   |

**(16pts) 1. Short answer (2 points each)**

**(2 pts) (1.1)** Of all the available datapath components, please specify (place a checkmark) whether the component will be needed for the execution of instruction: `addi x10, x12, 1`

Yes      NO

- |                                     |                                     |                           |
|-------------------------------------|-------------------------------------|---------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Instruction memory        |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | ALU                       |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Register file             |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Immediate generation unit |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | Data memory               |

**(2 pts) (1.2)** For each of the following statement, place a checkmark in the column identifying which memory technology, SRAM or DRAM, the statement best describes.

SRAM

DRAM

- |                                     |                                     |                                            |
|-------------------------------------|-------------------------------------|--------------------------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | is used for cache memory                   |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | is used for main memory                    |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | is the faster one of the two technologies  |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | is the cheaper one of the two technologies |

**(2 pts) (1.3).** Computer architect divide processors based on design criteria of instruction sets into two broad categories: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Choose **all** the following characteristics that **are applicable to RISC processor**.

- a. Variable-length instructions
- ☒ b. Fixed-length instructions
- c. Many instructions (typically, hundreds)
- ☒ d. Strives for a minimum instruction set
- ☒ e. Each instruction performs a basic computation
- f. Each instruction can perform an arbitrarily complex computation

**(2 pts) (1.4)** Please describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

HLL  $\xrightarrow{\text{Assembler}}$  Assembly  $\xrightarrow{\text{Compiler}}$  Machine

**(2 pts) (1.5).** Assume that one processor has **13 pipeline stages**, assume it is perfectly pipelined and the program is very long, how much speed up will it achieve compared to a nonpipelined processor.

Answer speed up is 13 times

**(2 pts) (1.6).** Derived from the paper-and-pencil method learned in grammar school, integer multiplication in computer is accomplished by iterative shifting and adding.



2 (2 pts) (1.7). The single-cycle datapath conceptually described in the class and in the textbook must have separate **Instruction and Data memories** because (single choice)

- a. The format of data and instructions is different in RISC-V and hence different memory are needed.
- b. having separate memories is less expensive.
- c. The processor operate in one cycle and cannot use a single-ported memory for two different accesses within that cycle.

2 (2 pts) (1.8). Name the **special purpose register** that stores the **address** of the next instruction to be fetched from memory.

Answer: PC (Program Counter)

15.5 (16pts) 2. Concepts Understanding (4 points each)

4 (2.1) There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards. Please name what are the **three different types of hazards**?

Structure Hazards  
Control Hazards  
Data Hazards

3.5 (2.2). It is the principle of locality that gives us a chance to overcome the long latency of memory access. Please explain what are **temporal locality** and **spacial locality**, and give an **very simple example** (in the area of computer architecture) to illustrate the point.

Ex Locality is used to develop a pattern of memory accesses.  
(1), AC(100) Temporal locality - The concept that a resource that is referenced at one point will be referred again in the future  
(1), AC(1), AC(200) Spacial locality - concept referencing a resource is higher if a resource near that was just referenced

(1), AC(2) (2.3) When determine the CPU performance, CPU time is related to Instruction count, CPI and clock cycle. Please fill out the following table to check which aspects are affected by the specific hardware and software component.

|                      | Instruction count | CPI | Clock cycle |
|----------------------|-------------------|-----|-------------|
| Algorithm            | X                 | X   |             |
| ISA                  | X                 | X   | X           |
| Compiler             | X                 | X   |             |
| Programming language | X                 | X   |             |

(2.4) Could you briefly explain/demonstrate Memory Hierarchy in a typical PC setting.

4 Memory Hierarchy - separates computer storage into a hierarchy based on response time

5 hierarchies: Cache, Main memory, magnetic discs and magnetic tapes

(12 pts) 3 Assuming an Instruction cache miss rate for a program is 1%, and a Data cache miss rate is 5%. If a processor has a ideal CPI of 2 without any memory stalls, and the miss penalty is 100 cycles for all misses. Assume that: 30% of the instructions are load & stores, 50% of the instructions are ALUs, and 20% of the instructions are controls.

(6 pts) 3.1 Data cache =  $1 * (0.30) * (0.05) * 100 = 1.5$

(i) What is actual CPI with the possible cache misses?

(ii) Assume that we have perfect data and instruction cache (no cache misses), what is the performance speed up of the CPU with the ideal perfect cache vs. the CPU with actual cache.

i) CPI (Clock Cycle Per Instruction)

$$\text{Actual CPI} = (\text{CPI}(\text{Ideal}) + 1 \text{ cache} + 1 \text{ Data cache})$$

$$= (2 + 1 + 0.15) = 3.15$$

ii) CPU Performance = Actual CPI / Ideal CPI

$$3.15 / 2 = 1.575 \text{ faster}$$

(6 pts) 3.2 What if the processor is faster, but memory system is not. Suppose we speed up the ideal CPI without any memory stalls from 2 to 1 without changing the clock rate.

(i) What is the CPI of the system with cache misses?

(ii) What is the performance speed up of the CPU with the ideal perfect cache vs. the CPU with actual cache?

(iii) What is your observation on the effect of the memory on the system performance?

$$\text{i) Actual CPI} = (\text{CPI}(\text{Ideal}) + 1 \text{ cache} + 1 \text{ Data cache})$$

$$= (1 + 1 + 0.15) = 2.15$$

$$\text{ii) CPU Performance} = \text{Actual CPI} / \text{Ideal CPI}$$

$$= 2.15 / 1 = 2.15 \text{ faster}$$

iii) Case 2 is faster, so the ideal CPI without any memory stalls decreases then system performance increases.

15

(20pts) 4.

(4 pts) (4.1) Given a 32-bit binary number, please demonstrate the range it could represent if it represent a single-precision IEEE 754 floating-point number.

Signed

unsigned

$$\left[ -2^{n-1}, -2^{n-1} - 1 \right] = \left[ -2^{31}, -2^{31} - 1 \right] \quad \left[ 0, 2^n - 1 \right] = \left[ 0, 2^{32} - 1 \right]$$

4

(4 pts) (4.2). What is overflow and underflow for floating point arithmetic?

overflow - A situation in which a positive exponent becomes too long to fit in the exponent field

underflow - A situation in which a negative exponent becomes too large to fit in the exponent field

6

(6 pts) (4.3) What is the decimal value of this IEEE 754 single-precision floating point number

0111 0001 1010 1110 1100 0000 0000 0000

|                |                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 2 <sup>8</sup> | 2 <sup>7</sup> | 2 <sup>6</sup> | 2 <sup>5</sup> | 2 <sup>4</sup> | 2 <sup>3</sup> | 2 <sup>2</sup> | 2 <sup>1</sup> | 2 <sup>0</sup> |
| 256            | 128            | 64             | 32             | 16             | 8              | 4              | 2              | 1              |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |

S = 0

Fraction = 01011101100<sub>2</sub>

Exponent = 11100011<sub>2</sub> = 227

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

$$x = (-1)^0 \times (1 + 0.010111011) \times 2^{227 - 127}$$

5

(6 pts) (4.4) Please represent the decimal number: -188.875 using single-precision floating point IEEE 754 standard.

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

$$(-1)^1 \times (1 + 10111100111) \times 2^{(134 - 127)}$$

188  
-128  
60  
-32  
28  
-16  
12  
-8  
4  
-4  
0

188 = 10111100

.875 = .111

188.875 = 10111100111

E - 127 = 7

E = 134

|                |                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 2 <sup>8</sup> | 2 <sup>7</sup> | 2 <sup>6</sup> | 2 <sup>5</sup> | 2 <sup>4</sup> | 2 <sup>3</sup> | 2 <sup>2</sup> | 2 <sup>1</sup> | 2 <sup>0</sup> |
| 256            | 128            | 64             | 32             | 16             | 8              | 4              | 2              | 1              |
| 0              | 1              | 0              | 1              | 1              | 1              | 1              | 0              | 0              |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |
|                |                |                |                |                |                |                |                |                |

$$2^{-1} = 0.5$$

$$2^{-2} = 0.25$$

$$2^{-3} = 0.125$$

$$2^{-4} = 0.0625$$

$$134 = 10000110$$

134  
-128  
6  
-4  
2  
-2  
0

11000 0110 10111111 000000

S Exponent

Fraction

-5-

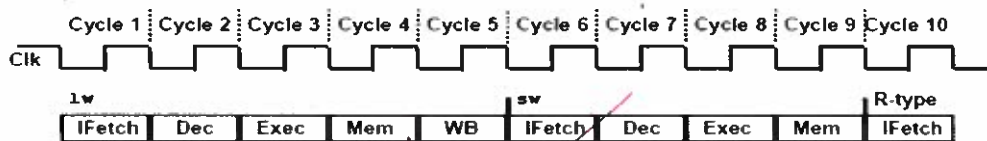
**(18 pts) 5. Single cycle implementation – multi cycle implementation – pipelined implementation:**

Calculate cycle time assuming negligible delays (for muxes, control unit, immediate Gen, PC access, shift left 1, wires) except: Instruction and Data Memory (200ps); ALU and adders (200ps); and Register File access (reads or writes) (100 ps)

**(5 pts) (5.1)** (i) Please fill in the table and calculate the total time needed for each instruction  
(ii) What will be a good time for a single cycle time (2pts)?

| Instruction Class    | Instruction memory | Register Read | ALU Operation | Data Mem | Register Write | Total |
|----------------------|--------------------|---------------|---------------|----------|----------------|-------|
| R type( such as ADD) | 200ps              | 100ps         | 200ps         |          | 100ps          | 600ps |
| Load (lw)            | 200ps              | 100ps         | 200ps         | 200ps    | 100ps          | 800ps |
| Store (sw)           | 200ps              | 100ps         | 200ps         | 200ps    |                | 700ps |
| BEQ                  | 200ps              | 100ps         | 200ps         |          |                | 500ps |

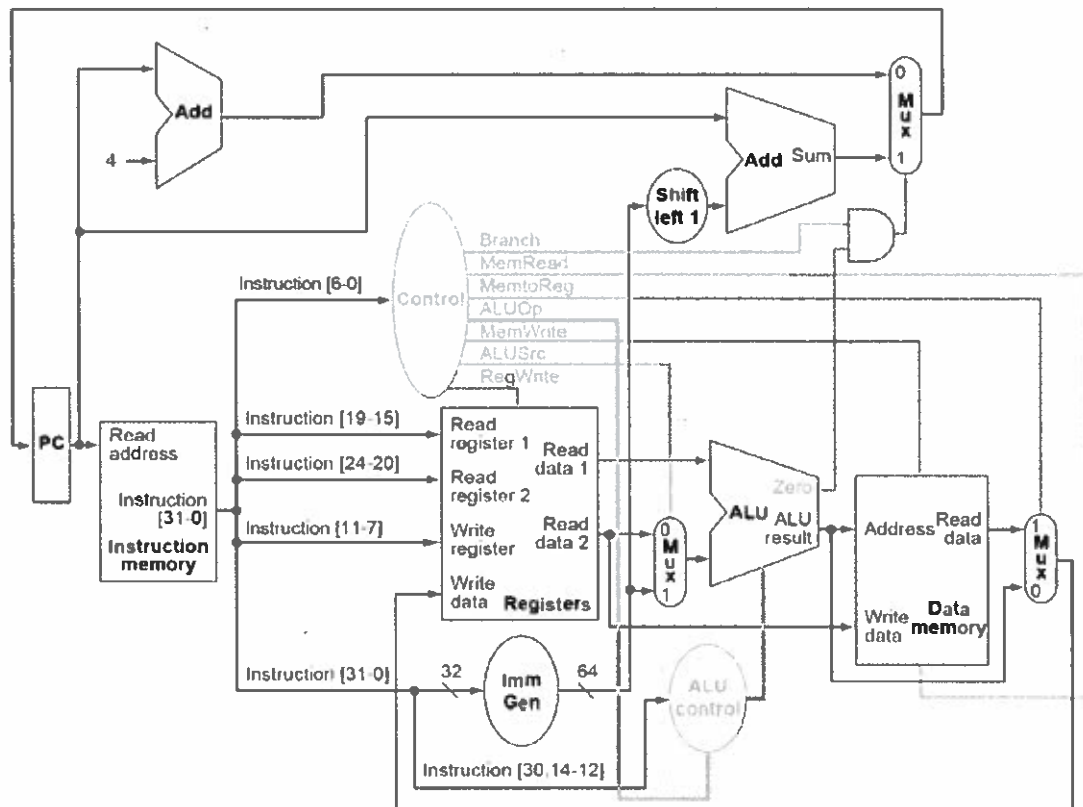
**(2 pts) (5.2)** when we adopt a multicycle implementation. Each step will take 1 clock cycle, please explain what will be an appropriate cycle time?



200ps - Highest time taken to perform any step

**(3 pts) (5.3)** Please explain what is the pipelined implementation.

Pipelining - Runs program faster by overlapping the execution of instructions



(4 pts) 5.4. Please fill in the following table, which specify how to set the control signals when executing the 2 instructions.

| Instruction | Branch | MemtoReg | ALUSrc | RegWrite | MemRead | MemWrite | ALUOp |
|-------------|--------|----------|--------|----------|---------|----------|-------|
| ld          | 0      | 1        | 1      | 1        | 1       | 0        | 00    |
| Beq         | 1      | X        | 0      | 0        | 0       | 0        | 01    |

(4 pts) 5.5. Please explain the detailed execution steps for the instruction:

sd x10, 40(x11)

SD (store doubleword) - purpose is to store double word to memory  
 Format: SD rt, offset (base)  
 x10 is GPR (general purpose register)  
 x11 is base, 40 is offset

The instruction saves the content of rt to the memory which is (base + offset). The content of x10 transfers to location memory (x11 + 40) (where x11 holds the base address of the memory location) and the 16-bit signal offset is added to the content of GPR base (x11) to form the effective address in the memory.

(10 pts) 6. The following RISC-V code segment is corresponding to the C code:  
 $A = B - E; \quad C = B + F;$

1. ld x1, 0(x6)  
 2. ld x2, 8(x6)  
 3. sub x3, x1, x2  
 4. sd x3, 16(x6)  
 5. ld x4, 24(x6)  
 6. add x5, x1, x4  
 7. sd x5, 40(x6)

(4 pts) 6.1 Assume forwarding (aka bypassing) scheme is adopted here. (i) Please specify the data hazards even with the forwarding scheme and point out where stalls should be added. (ii) how many clock cycles do you need to implement the above code sequence?

4 + 7 = 11 clock cycles

Plus stalls 11 + 2 = 13 clock cycles

(4pts) 6.2 (i) Please reorder the code segment (while do not change the function of the program) to avoid the stalls. (ii) how many clock cycles do you need to implement the above code sequence?

1. ld x1, 0(x6)      5. sd x3, 16(x6)  
 2. ld x2, 8(x6)      6. add x5, x1, x4  
 3. ld x4, 24(x6)  
 4. sub x3, x1, x2      7. sd x5, 40(x6)

(2pts) 6.3 What is the minimum number of cycles needed to completely execute N instructions on a CPU with a K stage pipeline? Justify your formula.

$$K = N + (M + 1)$$

(8 pts) 7. Please change the following C statement to RISC-V statements.

Assume that base address of array A is in register x10, base address of array B is in register x11, and each array element is a doubleword (64 bit). Variables register allocation are:

c: x6, i: x7, and j: x8.

What is the RISC-V assembly code for the C statement?

$c = A[i] - B[j]$

A[i] { slli x6, x10, 3  
 add x6, x6, x7  
 ld x13, 0(x6)  
 B[j] { slli x6, x11, 3  
 add x6, x6, x8  
 ld x17, 0(x6)  
 sub x6, x13, x17

**BELOW AVERAGE**

**Below Average Student Original Graded Assignment**



**ELEG 4339 Computer Organization & Design**  
**Department of Electrical & Computer Engineering**  
**PVAMU**

Final Exam

Spring 2022

Name: (Print) Nasir Patel  
(Sign) Nasir Patel

|              |     |     |
|--------------|-----|-----|
| 1            | 16  | 12  |
| 2            | 16  | 14  |
| 3            | 12  | 1   |
| 4            | 20  | 15  |
| 5            | 18  | 17  |
| 6            | 10  | 9.5 |
| 7            | 8   | 7.5 |
| <b>Total</b> | 100 | 76  |

12  
1.5  
(16pts) 1. Short answer (2 points each)

(2 pts) (1.1) Of all the available datapath components, please specify (place a checkmark) whether the component will be needed for the execution of instruction: `addi x10, x12, 1`

| Yes                                 | NO                                  |                           |
|-------------------------------------|-------------------------------------|---------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Instruction memory        |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | ALU                       |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | Register file             |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | Immediate generation unit |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | Data memory               |

2  
(2 pts) (1.2) For each of the following statement, place a checkmark in the column identifying which memory technology, SRAM or DRAM, the statement best describes.

| SRAM                                | DRAM                                |                                            |
|-------------------------------------|-------------------------------------|--------------------------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | is used for cache memory                   |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | is used for main memory                    |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | is the faster one of the two technologies  |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | is the cheaper one of the two technologies |

2  
(2 pts) (1.3). Computer architect divide processors based on design criteria of instruction sets into two broad categories: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). Choose **all** the following characteristics that **are applicable to RISC processor**.

- a. Variable-length instructions
- ☒ b. Fixed-length instructions
- c. Many instructions (typically, hundreds)
- ☒ d. Strives for a minimum instruction set
- ☒ e. Each instruction performs a basic computation
- f. Each instruction can perform an arbitrarily complex computation

1.5  
(2 pts) (1.4) Please describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

HLL  $\xrightarrow{\text{Assembler}}$  Assembly language  $\xrightarrow{\text{Compiler}}$  machine language

0  
(2 pts) (1.5). Assume that one processor has **13 pipeline stages**, assume it is perfectly pipelined and the program is very long, how much speed up will it achieve compared to a nonpipelined processor.

Answer 13 times

1  
(2 pts) (1.6). Derived from the paper-and-pencil method learned in grammar school, integer multiplication in computer is accomplished by iterative addition and shift.

(2 pts) (1.7). The single-cycle datapath conceptually described in the class and in the textbook must have separate **Instruction and Data memories** because (single choice)

a. The format of data and instructions is different in RISC-V and hence different memory are needed.

b. having separate memories is less expensive.

c. The processor operate in one cycle and cannot use a single-ported memory for two different accesses within that cycle.

(2 pts) (1.8). Name the **special purpose register** that stores the **address** of the next instruction to be fetched from memory.

Answer: PC

(16pts) 2. Concepts Understanding (4 points each)

(2.1) There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards. Please name what are the **three different types of hazards**?

Data hazard - need to wait for previous instruction to complete its data read/write

Structure hazard - A required source is busy

control hazard - Deciding on control action depends on previous instruction

(2.2). It is the principle of locality that gives us a chance to overcome the long latency of memory access. Please explain what are **temporal locality** and **spacial locality**, and give an very **simple example** (in the area of computer architecture) to illustrate the point.

temporal locality - States that if data location is referenced then it will be referenced again, for example you are already in a data memory and you will go there again to store something else

Spacial locality - States that if data is referenced then the data location nearby will be referenced, for example you to an address you will store the result nearby

(2.3) When determine the CPU performance, CPU time is related to Instruction count, CPI and clock cycle. Please fill out the following table to check which aspects are affected by the specific hardware and software component.

|                      | Instruction count | CPI | Clock cycle |
|----------------------|-------------------|-----|-------------|
| Algorithm            | ✓                 | ✓   |             |
| ISA                  | ✓                 | ✓   | ✓           |
| Compiler             | ✓                 | ✓   |             |
| Programming language | ✓                 | ✓   |             |

(2.4) Could you briefly explain/demonstrate Memory Hierarchy in a typical PC setting.

Memory hierarchy is where there are levels of importance in memory or where memory is needed in different times while trying to execute a code

(12 pts) 3 Assuming an **Instruction cache miss rate for a program is 1%**, and a **Data cache miss rate is 5%**. If a processor has a **ideal CPI of 2** without any memory stalls, and the **miss penalty is 100 cycles for all misses**. Assume that: 30% of the instructions are load & stores, 50% of the instructions are ALUs, and 20% of the instructions are controls.

(6 pts) 3.1

- (i) What is actual CPI with the possible cache misses?
- (ii) Assume that we have perfect data and instruction cache (no cache misses), what is the **performance speed up** of the CPU with the ideal perfect cache vs. the CPU with actual cache.

Instruction cach = 1% miss penalty = 100

Data cach = 5% ideal CPI = 2

$$i) CPI = 2 + ($$

(6 pts) 3.2 What if the processor is faster, but memory system is not. **Suppose we speed up the ideal CPI without any memory stalls from 2 to 1** without changing the clock rate.

- (i) What is the CPI of the system with cache misses?
- (ii) What is the performance speed up of the CPU with the ideal perfect cache vs. the CPU with actual cache?
- (iii) What is your observation on the effect of the memory on the system performance?

15  
♡

(20pts) 4.

(4 pts) (4.1) Given a 32-bit binary number, please demonstrate **the range** it could represent if it represent a single-precision IEEE 754 floating-point number.

$$[0, 2^{32}-1] \quad [-2^{31}, 2^{31}-1]$$

3

(4 pts) (4.2). What is overflow and underflow for floating point arithmetic?

Overflow - A situation where a positive integer can't fit in an exponent field

Underflow - A situation where a negative integer can't fit in the integer field

(6 pts) (4.3) What is the decimal value of this IEEE 754 single-precision floating point number

6

0111 0001 1010 1110 1100 0000 0000 0000

$$227 - 127 = 100$$

$$1 * 1.01011101100 \times 2^{100}$$

6

(6 pts) (4.4) Please represent the decimal number: **-188.875** using single-precision floating point IEEE 754 standard.

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

$$\begin{aligned} 0.875 \times 2 &= 1.75 \\ 0.75 \times 2 &= 1.5 \\ 0.5 \times 2 &= 1.0 \end{aligned}$$

$$\begin{aligned} 188/2 &= 94 \\ 94/2 &= 47 \\ 47/2 &= 23 \\ 23/2 &= 11 \\ 11/2 &= 5 \\ 5/2 &= 2 \\ 2/2 &= 1 \\ 1/2 &= 1 \end{aligned}$$

$$\begin{aligned} &10111100.111 \\ &2^7 \\ &E - 127 = 7 \\ &\quad +127 \quad +127 \\ &\hline &E = 134 \end{aligned}$$

$$1100001101011100111\dots$$

-5-

(18 pts) 5. Single cycle implementation – multi cycle implementation – pipelined implementation:

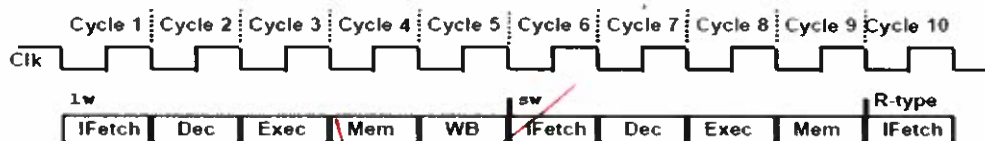
Calculate cycle time assuming negligible delays (for muxes, control unit, immediate Gen, PC access, shift left 1, wires) except: Instruction and Data Memory (200ps); ALU and adders (200ps); and Register File access (reads or writes) (100 ps)

- (5 pts) (5.1) (i) Please fill in the table and calculate the total time needed for each instruction  
(ii) What will be a good time for a single cycle time (2pts)?

| Instruction Class    | Instruction memory | Register Read | ALU Operation | Data Mem | Register Write | Total |
|----------------------|--------------------|---------------|---------------|----------|----------------|-------|
| R type( such as ADD) | 200ps              | 100ps         | 200ps         | X        | 100ps          | 600ps |
| Load (lw)            | 200ps              | 100ps         | 200ps         | 200ps    | 100ps          | 800ps |
| Store (sw)           | 200ps              | 100ps         | 200ps         | 200ps    | X              | 700ps |
| BEQ                  | 200ps              | 100ps         | 200ps         | X        | X              | 500ps |

ii) A good single cycle time would be 800ps

- (2 pts) (5.2) when we adopt a multicycle implementation. Each step will take 1 clock cycle, please explain what will be an appropriate cycle time?

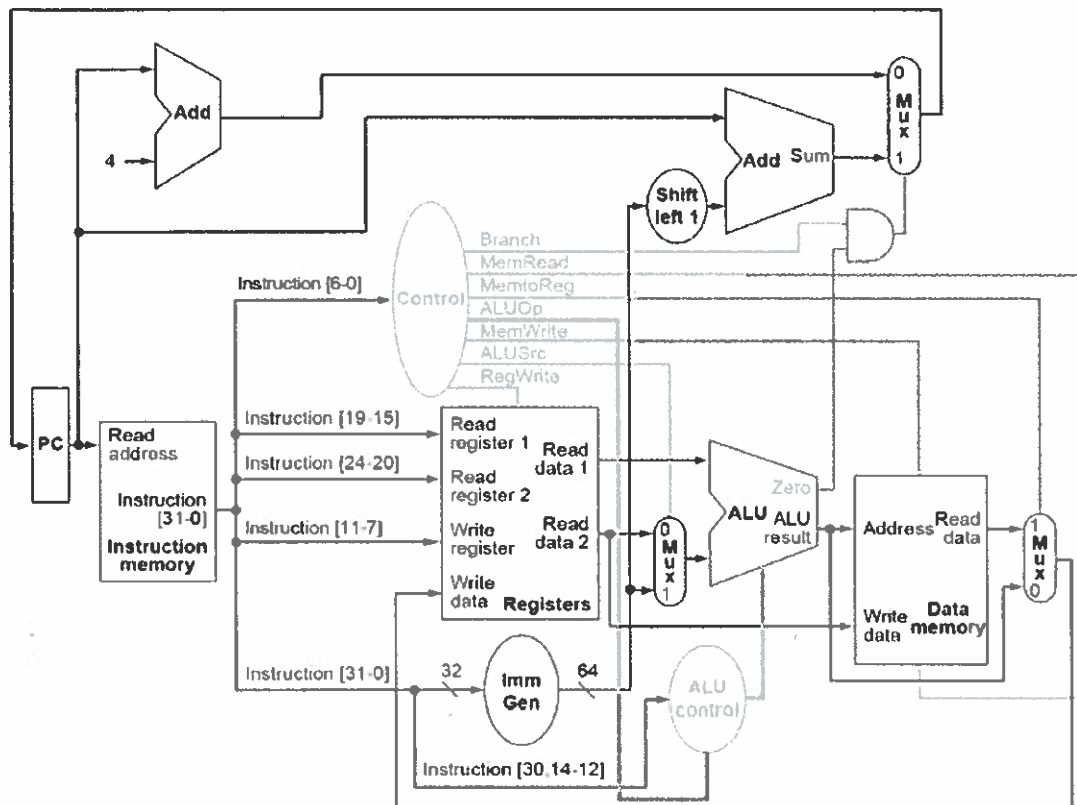


- The appropriate cycle time will be 200ps

- (3 pts) (5.3) Please explain what is the pipelined implementation.

- Pipelined implementation is overlapping instruction to improve instruction throughput





(4 pts) 5.4. Please fill in the following table, which specify how to set the control signals when executing the 2 instructions.

| Instruction | Branch | MemtoReg | ALUSrc | RegWrite | MemRead | MemWrite | ALUOp |
|-------------|--------|----------|--------|----------|---------|----------|-------|
| ld          | 0      | 1        | 1      | 1        | 1       | 0        | 00    |
| Beq         | 1      | X        | 0      | 0        | 0       | 0        | 01    |

(4 pts) 5.5. Please explain the detailed execution steps for the instruction:

sd x10, 40(x11)

1. Fetch the instruction from instruction memory
2. Register x11 value is written to the offset
3. The ALU adds the offset to register ~~x10~~
4. The sum of the ALU is used as address for data memory
5. Data is stored in reg file ~~No~~

**(10 pts)** 6. The following RISC-V code segment is corresponding to the C code:  
 $A = B - E; \quad C = B + F;$

1. ld x1, 0(x6)  
 2. ld x2, 8(x6)  
 3. sub x3, x1, x2  
 4. sd x3, 16(x6)  
 5. ld x4, 24(x6)  
 6. add x5, x1, x4  
 7. sd x5, 40(x6)

i) ld x1, 0(x6)  
 ld x2, 8(x6)  
 ld x4, 24(x6)  
 sub x3, x1, x2  
 sd x3, 16(x6)  
 add x5, x1, x4  
 sd x5, 40(x6)

**(4 pts)** 6.1 Assume forwarding (aka bypassing) scheme is adopted here. (i) Please specify the data hazards even with the forwarding scheme and point out where stalls should be added. (ii) how many clock cycles do you need to implement the above code sequence?

i) The data hazard needs to wait for previous instruction to complete its data read/write and the hazards are at

ii) 13 clock cycles

**(4 pts)** 6.2 (i) Please reorder the code segment (while do not change the function of the program) to avoid the stalls. (ii) how many clock cycles do you need to implement the above code sequence?

ii) 11 clock cycles

**(2 pts)** 6.3 What is the minimum number of cycles needed to completely execute N instructions on a CPU with a K stage pipeline? Justify your formula.

$$= N + (K - 1)$$

**(8 pts)** 7. Please change the following C statement to RISC-V statements.

Assume that base address of array A is in register x10, base address of array B is in register x11, and each array element is a doubleword (64 bit). Variables register allocation are:

c: x6, i: x7, and j: x8.

What is the RISC-V assembly code for the C statement?

$c = A[i] - B[j]$

slli x7, x7, 3

add x10, x10, x7

ld x7, 0(x10)

slli x8, x8, 3

add x11, x11, x8

ld x8, 0(x11)

sub x6, x7, x8

Register allocation



# **ELEG 4339-P01 COMPUTER ORGANIZATION & DESIGN**

**SPRING 2022**

**Title of Assignment: Project 2**

## **ELEG4339 Computer Architecture & Organization**

**Spring 2022**

### **Project 2: Modern Processor Design.**

One of the main objectives of this course is to understand the modern computer architecture and organization. Throughout this course, the computer performance concept is introduced in chapter one. Computer performance in the perspective of CPU time is determined by three key factors: instruction count, clock cycle time, and the CPI. In chapter 2 and 3, we explained that the compiler and some details about Instruction set architecture using RISC-V ISA as an example. In chapter 4, the principle and the techniques used in implementing a processor, starting with a highly abstract and simplified single-cycle implementation to explained the concepts of Datapath components and Control. Then we moved on to a more realistic pipelined multi-cycle RISC-V implementation. Memory system design will be introduced in chapter 5. The subject of modern processor design is ever evolving. In this project, You will need to identify, retrieve, and organize information about the modern processor design, based on the material we have learned in the class, and at the same time taking advantage of learning opportunities available in the online resources, technical papers, and other sources to gain the new knowledge in the area of the modern processor design. In this project, you should develop a study plan, and discuss the appropriate learning strategies to acquire new knowledge to better understand the subject.

This project will account for **15%** towards your final grade. In this project, please include the following components.

#### **Part I: The discussion of different learning strategies. (20%)**

Please complete the on-line questionnaire. “Index of learning styles questionnaire” (<https://www.engr.ncsu.edu/ils/>), and read the paper “Learning and teaching styles in engineering education”. Please do not limit yourself in this source. In this project, you may want to explore different learning strategies based on your experience and research papers in the literature, especially in the context of Electrical and Computer engineering.

#### **Part 2: Please use the handout/slides/chapter 4 as a guideline. Explain the principle and techniques used in implementing a processor. (30%)**

For example, from single cycle implementation (builds up a datapath and constructs a simple version of a processor sufficient to implement an instruction set like RISC-V) – multi cycle implementation – pipelined implementation (a more realistic implementation).

#### **Part3: Please use a real processor as a case study to explore the topic of modern processor design (50%).**

Developed from pipelining, modern processors are trying to exploit the potential parallelism among instructions. This parallelism is called, instruction-level parallelism (ILP). There are two primary ways to increase the potential ILP. The first is increasing the depth of the pipeline to overlap more instructions. The second is replicate components of the computer so that it can launch multiple instructions in every pipeline state (multiple issue). There are other topics in modern processor design to improve the performance. Please use book, papers, company websites, etc. to study this topic. Please use some modern processors as case study.

- (a) choice 1: The ARM Cortex-A53
- (b) choice 2: Intel Core i7
- (c) choice 3: apple M1, M1 ultra
- (d) choice 4: AMD processors, such as Ryzen series
- (e) choice 4: NVIDIA processors, such as the Grace CPU Superchip recently released
- (f) You could also research and pick other modern processors.

Please make sure to find additional resources to explore this topic and make sure to include the references in your report. At the same time, please discuss and demonstrate the appropriate use of the new knowledge to solve problems.

**This project is a team project, up to 3 students in each team. Both reports and presentation are expected from this project.**

**Deliverable:**

- (1) **Each student** should submit a report to eCourses (**Outcome 7 – An ability to acquire and apply new knowledge as needed, using appropriate learning strategies**). **The report should cover part one, two, and three. Part 2&3 could be the same for each team, but part 1 should be specific to each student.**
- (2) A “mini-conference” will be offered in class (Apr. 28<sup>th</sup>). Each team should prepare about 15-20 minutes presentation mainly about part 3. The presentation slides should be submitted to eCourses. The tentative schedule for presentation is Apr. 28<sup>th</sup>.

**Due date: Both report and presentation should be submitted before Apr. 28th.**

**This project will account for 15% towards your final grade. Both report and presentation will be graded.**

## **Students' Project Report**

**ABOVE AVERAGE**

**Above Average Student Original Graded Assignment**

**ARM Cortex A-53 Processor**  
**ELEG 4339**  
**Computer Architecture and Organization**



Dr. Li  
Xavier Dukes  
Juwan Byers  
Oluwadamilola Osuntunyi  
04/28/2022

85

## Discussion of Different Learning Styles

### Juwan Byers

My results for my learning strategy according to the North Carolina State University Index of Learning Styles can be viewed in Figure 1.

Questionnaire Results for Juwan Byers

➤ Active 3 ➤ Sensing 7 ➤ Visual 9 ➤ Sequential 3



**Figure 1**

According to NCSU ILS, "If your score for a dimension is 1 or 3, you are fairly well balanced on the two categories of that dimension, with only a mild preference for one or the other. If your score for a dimension is 5 or 7, you have a moderate preference for one category of that dimension. You may learn less easily in an environment that fails to address that preference at least some of the time than you would in a more balanced environment. If your score for a dimension is 9 or 11, you have a strong preference for one category of that dimension. You may have difficulty learning in an environment that fails to address that preference at least some of the time." For the categories Active/Reflective and Sequential/Global, I scored a three which means that I am balanced and have a preference of Active in the Active/Reflective category and Sequential in the Sequential/Global category. Active learners tend to retain and understand information best by doing something active with it—discussing or applying it or explaining it to others. Reflective learners prefer to think about it quietly first (Felder and Solomon, 1993). Sequential learners tend to follow logical stepwise paths in finding solutions; global learners may be able to solve complex problems quickly or put things together in novel ways once they have grasped the big picture, but they may have difficulty explaining how they did it (Felder and Solomon, 1993). Further, I scored a seven for Sensing in the Sensing/Intuitive category. A seven represents that I have a preference for Sensing in the Sensing/Intuitive category. One of the facts about Sensors is that they often like solving problems by well-established methods and dislike complications and surprises (Felder and Solomon, 1993). When the opportunity presents itself a google search of a question that pops in my mind is one of my favorite leisure activities and leads me to go down the rabbit hole. Lastly, a number nine was received from the Visual side of the Visual/Verbal category. Visual learners remember best what they see—pictures, diagrams, flow charts, time lines, films, and demonstrations (Felder and Solomon, 1993). My best tool for

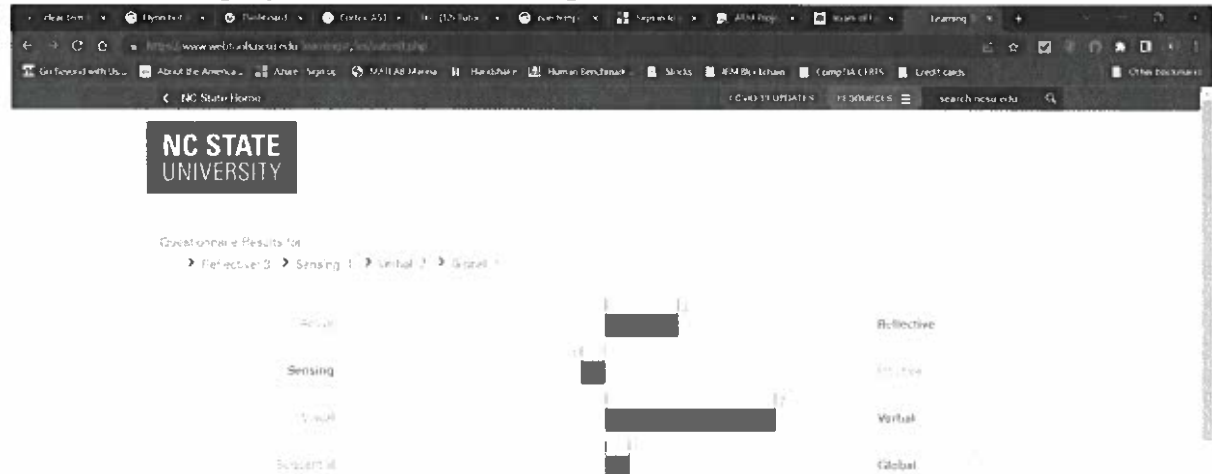


learning is watching YouTube videos or book examples of the problem being completed for me to learn the material. Also, I can learn from doing the activity such as fixing my car because once I see it one time I can remember what was done and repeat the process.

*personalized learning style*

## Xavier Dukes

My results for my learning strategy according to the North Carolina State University Index of Learning Styles can be viewed in Figure 2.



What do my results mean?



Figure 2

I personally have a strong tendency towards verbal and auditory forms of learning. It is stated that "Verbal learners get more out of words—written and spoken explanations" (Felder). I have also enjoyed lectures over many other forms of learning such as experiments or demonstrations. Not only do I prefer this form of learning I have experience in the past actually learning less effectively when presented with visual or graphic means to learn. The document titled "LEARNING STYLES AND STRATEGIES" does cover this predicament where it states that "You may learn less easily in an environment that fails to address that preference at least some of the time than you would in a more balanced environment" (Felder). However despite this more one sided nature of my learning it is stated that a balance of both has been shown to be optimal in most cases "Everyone learns more when information is presented both visually and verbally" (Felder). Learning verbally can be supplemented via the writing of summaries or group work and the associated communication that comes with such work according to the Felder. I am interested in expanding my strategies to become a bit more balanced overall so that I can be an all around effective learner however, I can still maintain some preferences.

## Oluwadamilola Osuntuyi:

Questionnaire Results for **oluwadamilola osuntuyi**:

➤ Active: 7 ➤ Sensing: 5 ➤ Visual: 9 ➤ Sequential: 5

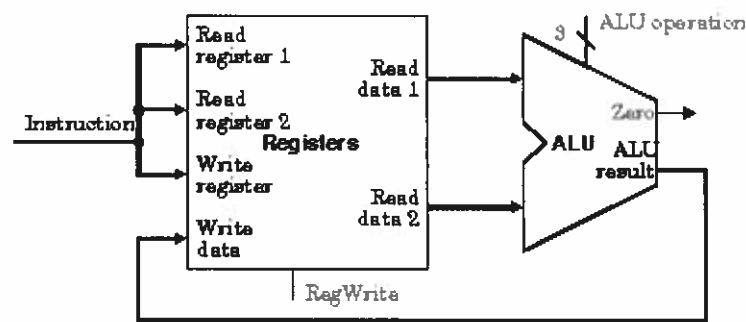


According to the model on which the ILS is based, there are four dimensions of learning style, with each dimension having two opposite categories. The reported score for a dimension indicates my preference for one category or the other. According to the result I am an Active, sensing, visual, and sequential. I very much agree with the results, according to the definitions. Active learners tend to retain and understand information best by doing something active with it, discussing or applying it or explaining it to others. Sensors tend to be patient with details and good at memorizing facts and doing hands-on (laboratory) work. Visual learners remember best what they see: pictures, diagrams, flow charts, time lines, films, and demonstrations. Sequential learners tend to gain understanding in linear steps, with each step following logically from the previous one. The results are not 100% accurate because sometimes I learn combining both dimension categories. I believe we tend to change the ways we learn as we grow.

## Principle and techniques used in implementing a processor

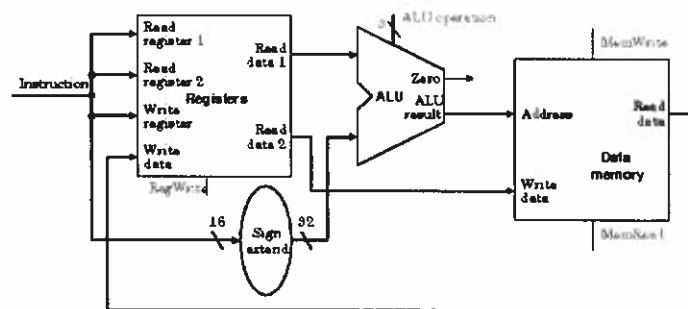
### Single-Cycle Implementation

The implementation schemes are different for the different type formats that are used such as R and I. For the R-format the register file and ALU are mainly used. The datapath taken and parts of hardware used are different for various instructions. For example, the instruction could be add x1,x2,x3 and the two register files x2 and x3 will be sent from the instruction memory to the read register 1 and 2. They will be sent from read data 1 and 2 as 32 bit numbers and added in the ALU and then written back in the write data as a 32 bit number and written in the write register x1 and a 5 bit number.



R-type datapath

For the I-type implementation, The Register file, ALU, data memory, and sign extend are used because of the multiplexers that are involved in the operation. The control unit has different instructions for the I format as it needs to send a signal from it to the data memory and a different ALUOp for the ALU operation. For this example, we will use lw x1,40(x3). The load instruction will be read by read register 1. Then the ALU will compute the sum of the register and the sign extended 12 bits of instruction. The sum is sent to the data memory and then sent to the write data which is written in the write register 5 bit file.

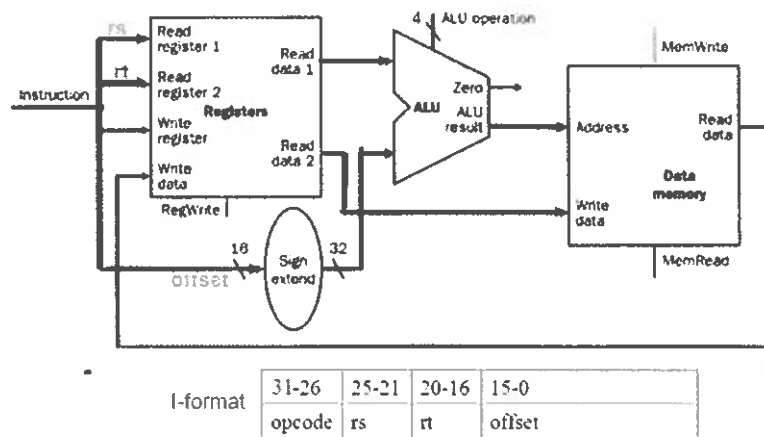


Load Word

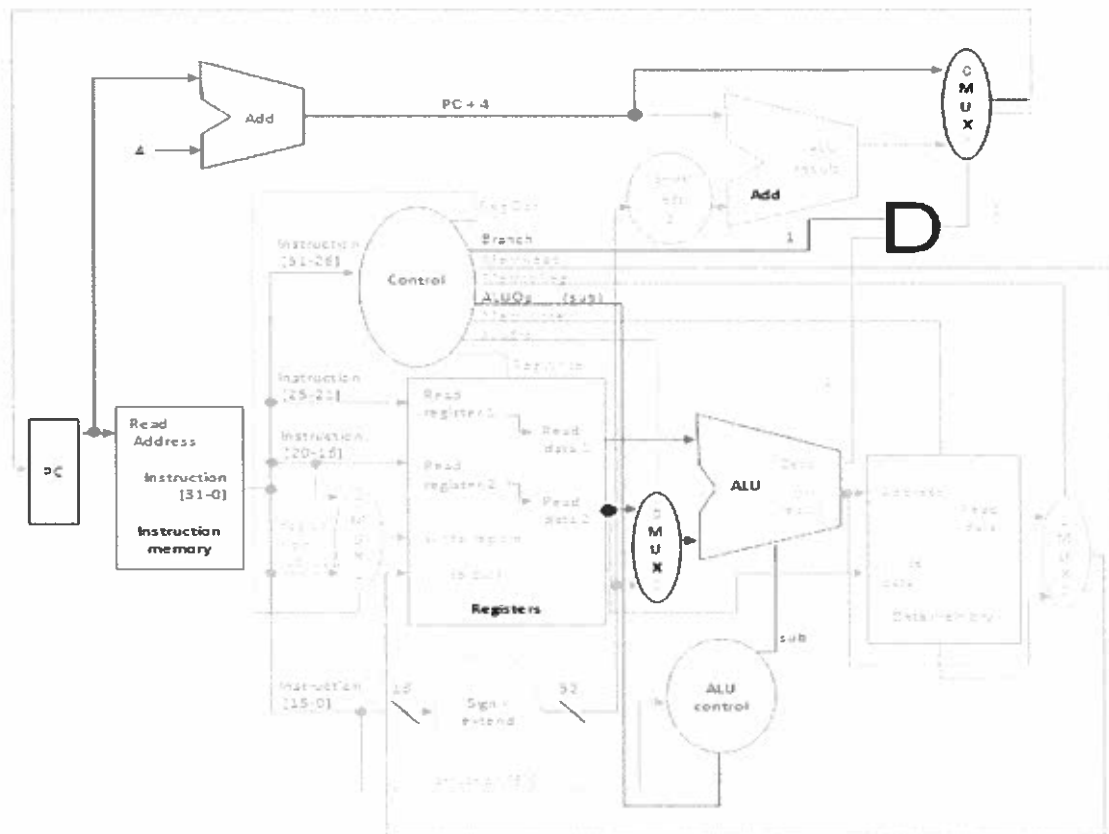
For the I-type implementation, The Register file, ALU, data memory, and sign extend are used because of the multiplexers that are involved in the operation. The control unit has different instructions for the I-format as it needs to send a signal from it to the data memory and a different ALUOp for the ALU operation. In the case for the store word, the ALU has the signal of 00 to add. For this example, we will use `sw x1,40(x3)`. The store instruction will be read by read register 1. Then the ALU will compute the sum of the register and the sign extended 12 bits of instruction. The sum is sent to the data memory and then the second register will be read and sent to the write data in data memory where the address will be stored until called upon.

## Store instruction datapath

`sw $t0, offset($t1): Mem[$t1 + se(offset)] = $t0`



For the SB-format, the instruction is gathered from the PC and sent to the instruction memory. The registers would be read and then sent to the ALU. The ALU then subtracts the two register values while the PC+4 value is sent to the multiplexer and the sign extended bit and the PC value is added and sent to the multiplexer. The value from the ALU will let the Zero what instruction to send to the multiplexer to perform and send back to the PC.



Branch Equal to

## Multi-Cycle/Pipeline Implementation

In more realistic scenarios, single cycle implementation is not feasible or efficient. A more realistic implementation of this would be a sort of multicycle implementation that makes use of short bursts of clock cycles that correlate to the stages of instruction execution. However, this stage by stage completion versus the single instruction implementation.

Pipelining is essential for max processor efficiency as it drastically increases the throughput of the processor, however during the implementation various complications can arise due to this rapid instruction fetching and execution. These complications or hazards can take the form of control, data, or structural hazards. A control hazard being a type of issue where the processor cannot fetch the next instruction due to a branch condition that has yet to be evaluated. A data hazard is typically caused by data not being accessible at the time it is needed by the next instruction because the previous may still be completing execution or hasn't written the value and stored it back in the register yet. Finally, a structural hazard is caused by several processes attempting to access the same resource at the same time causing one to wait or stall until the resource is free for use. All of these hazards slow the execution of the pipeline and keep it from being optimal. A general solution to all of these problems is to just stall the execution and create a "bubble" however this is the least efficient option as it is not maximizing the throughput, other options include hardware additions and connections or restructuring of code to prepare data well in advance before it is needed. In order to solve data hazards there are several viable solutions that can be implemented including forwarding or bypassing, code restructuring or reordering, and waiting. Forwarding is the act of creating additional hardware connections in order to provide data to the next instruction as soon as it is available from data memory or after the ALU operation depending on the instruction. This does not solve all instances of data hazards but does significantly reduce their effect on the CPI or IPC. Optimally ordered code paired with bypassing strategies and connections can virtually eliminate data hazards and make the implementation of the pipeline seamless in that respect. Structural hazards can be solved with some similar strategies such as code reordering, hardware redundancies, and waiting. Hardware redundancies allow for constant access to resources without waiting and code reordering can ensure that the times that these resources are accessed by various processes is spaced out enough to allow adequate time for completion with little to no wait times. Finally, control hazards are one of the hardest to deal with as there is an element of prediction and guessing that occurs as to whether or not branches are taken and what instruction is loaded and fetched next based on the conditional statement. The ways to solve control hazards include branch predictions, early calculation of conditional statements, and the preloading of instructions.

In conclusion, All of these methods are used to implement pipelines successfully and in their most optimal form in modern processors.

### Case Study: ARM Cortex A-53

The ARM Cortex A-53 processor is a 1-4 core, 64 bit, dual issue pipelined model of processor meaning it loads two instructions simultaneously and pipelines them for increased efficiency, this kind of dual issue implementation gives an Ideal CPI of 0.5 or IPC of 2. This microprocessor utilizes characteristics of typical RISC computers and was inspired by the Berkeley RISC machine that was researched back in 1983. The ARM also has a CPU and Floating point Processing Unit to handle various types of arithmetic. This processor has an 8-stage execution of instructions with a max clock rate of 2.65 GHZ. These 8 stages are as follows: instruction fetch, predict and register (which handles branch and jump predictions and early reads/writes), decode, execute, memory read, memory write, write back and finally the use of a forwarding stage that can make data available immediately available to the decoding stage of the next instruction. This use of forwarding or bypassing is a key aspect when implementing a pipeline and avoids data hazards that arise when attempting to maximize the throughput. This type of processor is very common in mobile devices and embedded systems such as the Raspberry Pi 3 and Zero. The ARMv8 style of processors that include the Cortex-A53 include microprocessors with over 26 million transistors per chip and we have seen significant growth from other ARM versions that in some cases have under 1 million.

This processor makes use of several features to increase CPI and makes the most of the multi step/stage clock cycles. It does this by limiting possible hazards and optimizing the pipeline with as little waiting time as possible. This processor utilizes the Harvard structure which is a computer architecture with separate storage and signal pathways for instructions and data, this helps ensure that structural hazards do not occur and therefore encroaches on the ideal CPI. This type of architecture allows for some redundancy and the freeing of various resources and therefore avoiding possible structural hazards. Some of these various redundancies include buses, L1 & L2 Data and instruction Caches for each core that is present. This processor also makes the common case fast as it optimizes its normal instructions or more frequently used ones and has specialized hardware to handle encryption or floating values via FPU.

Another hazard this processor is built to handle is the control hazard, it makes use of branch prediction and speculative instruction fetching which does not guarantee the execution of the instruction but functions as a sort of hit or miss opportunity and will have a pipeline flush depending on if the branch condition is true or false. The pipeline flush will dump the fetched instruction and begin execution of the proper instruction. Several other techniques that help handle control hazards include preloading instructions and data prefetching. Data prefetching is when "The data cache implements an automatic prefetcher that monitors cache misses in the core. When a pattern is detected, the automatic prefetcher starts linefills in the background. The prefetcher recognizes a sequence of data cache misses at a fixed stride pattern"(ARMDeveloper .2022). There is a Instruction Fetching Unit that sequentially grabs instructions from a 2 way cache and " obtains instructions .... and predicts the outcome of branches in the instruction



stream”(ARMDeveloper ,2022). This part of the processor helps limit control hazards with its prediction mechanism. The IFU will load instructions and then empty the one not used which is determined via branch condition,“This stores up to two instruction cache fetches and enables the branch shadow of predicted taken branch instructions to be eliminated”(ARMDeveloper ,2022). This predicted branch is stored in a 256 entry Branch Target Address Cache for quick access to all potential branch values, “the ARM Cortex A-53 processor hosts a 4 KiB conditional branch predictor, 256-entry indirect branch predictor”.

The ARM processor uses L2 and L1 Caches to store various instructions based off the frequency of their uses. Non-temporal instructions are stored in the level 2 cache by default as they are non frequently accessed instructions. Making room for the more common case instructions and therefore maximizing IPC.

In conclusion, there are many ingenious strategies that are implemented in order to optimize processors and increase IPC. The minimization of various types of hazards are essential in the creation of an ideal pipeline. Along with the dual issue nature of the processor there is potential for a CPI drop below 1. The Various ways that the processor limits the control, data, and structural hazards assists greatly in the implementation of the pipeline. The prediction, instruction preloading and various redundancies make up for the potential hazards and will minimize time needed for stalls. There is a key role played by the memory hierarchy as well that seeks to optimize clock cycles and timing when it comes to the frequent use of instructions through the IFU. Overall, this processor uses a myriad of strategies to maximize the throughput of the device.

*good summary !*



## Resources

Developer, A. R. M. (n.d.). Documentation – arm developer. Retrieved April 27, 2022, from <https://developer.arm.com/documentation/ddi0500/j/Level-1-Memory-System/Data-prefetching/Data-prefetching-and-monitoring?lang=en>

Developer, A. R. M. (n.d.). Cortex-A53. Retrieved April 27, 2022, from <https://developer.arm.com/Processors/Cortex-A53>

Felder , R., & Solomon, B. (1993). Resources for teaching and Learning Stem. Retrieved April 28, 2022, from <https://educationdesignsinc.com/index-of-learning-styles/>

Felder , R., & Solomon, B. (n.d.). *Learning styles and strategies1 Richard M. Felder ... - NCSU. LEARNING STYLES AND STRATEGIES*. Retrieved April 28, 2022, from <https://www.engr.ncsu.edu/wp-content/uploads/drive/1WPAfj3j5o5OuJMiHorJ-lv6fON1C8kCN/styles.pdf>

Ltd., A. (n.d.). *Cortex-A53*. Arm. Retrieved April 27, 2022, from <https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a53>

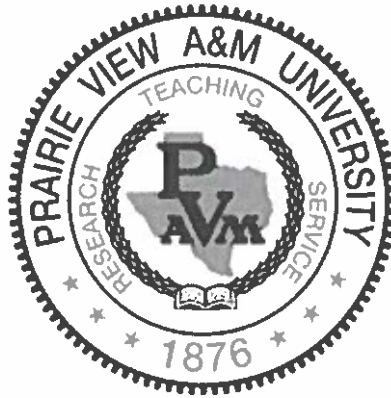
Schor, D. (2021, May 23). *ARM launches ARMv9*. WikiChip Fuse. Retrieved April 29, 2022, from <https://fuse.wikichip.org/news/4646/arm-launches-armv9/>

Wikimedia Foundation. (2022, April 6). *ARM cortex-A53*. Wikipedia. Retrieved April 28, 2022, from [https://en.wikipedia.org/wiki/ARM\\_Cortex-A53](https://en.wikipedia.org/wiki/ARM_Cortex-A53)

**AVERAGE**

## **Average Student Original Graded Assignment**

**Department of Electrical and Computer Engineering**  
**Prairie View A&M University**



**Project 2: Modern Processor Design**  
**NVIDIA-GRACE HOPPER SUPERCHIP**

**Computer Architecture & Organization**  
**ELEG 4339**

**Group Members:**

**Abel Mendoza, John Barrett, Jacqueline Torralba**

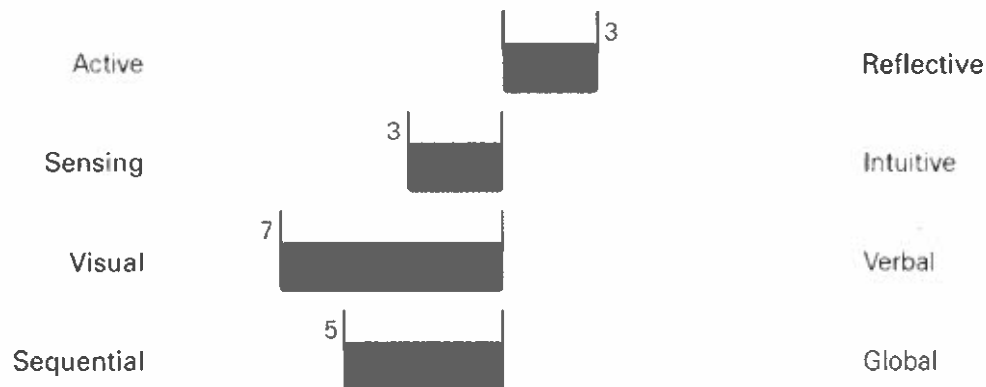
**Instructor: Dr. Li**

*80*

## Part 1: The discussion of different learning strategies

Questionnaire Results for John Barrett:

➤ Reflective: 3 ➤ Sensing: 3 ➤ Visual: 7 ➤ Sequential: 5



The way I went about learning and gathering information for the next to sections goes as follows. For Part 2 I tried to remember everything that I could on the subject. Then, I went back to look at the relevant information from what I could remember. For Part 3 I started by going to the website of the manufacturer, (NVIDIA). Finding all the info pertaining to the topic by searching the website. From what can be found on the website more searches can be made for relevant information. In this case all that could be found on the manufacturer website was one press release, as the CPU was only announced one month previously.

Part 2: Principle and techniques used in implementing a processor.

### Single-cycle implementation

In the single-cycle implementation, all implemented instructions are completed in one exact clock cycle. This exact clock cycle must be the same length for every instruction implementation. The longest instruction from the code, in most cases is the LW instruction, will determine the clock cycle for the whole implementation.

In the LW instruction case, it must go through the instruction memory, the register file, the ALU (arithmetic-logic unit), the data memory, and the register file. While compared to the ADD instruction which only goes through the instruction memory, the register file, the ALU, and the register file again, which is one less than the LW instruction. Despite the ADD instruction being shorter, when put through single-cycle implementation it will still take the same amount of clock cycle as if it were to be the LW instruction. This causes the overall performance of this technique to be poor, since the longest clock cycle is chosen and kept fixed.

## Multi-cycle implementation

In multicycle implementation, each step is executed in 1 clock cycle and will allow a functional unit to be used more than once per instruction if you are using it in another clock cycle. By doing this you end up reducing the amount of hardware required.

When comparing the multicycle implementation to single cycle implementation the difference will be:

- single memory unit is used both for instruction and data
- single ALU will be used rather than an ALU and two adders,
- one or more registers will be added to temporarily hold the value until the next subsequent clock cycle.

In a subsequent clock cycle, the data will be stored in a state element. If the data is subsequent instructions will be stored in the register file, the program counter (PC) or the memory.

While data used by the same instruction will be stored in additional registers that are attached to each functional unit. This leads to the following temporary registers to be added which are the

- Instruction registers, memory data register to save output of the memory for instruction read and data read.
- A & B registers to hold register operand values read from register file
- The ALUOut register that holds the output of the ALU

Below the multicycle data path is displayed in figure 1 that shows the different multiplexers used for selection. In the first multiplexer, the selection will be between the PC and ALUOut for memory address. The second multiplexer will be a selection between the destinations for register addresses. The third multiplexer will select between ALUOut and memory data for write data input to register file. The fourth multiplexer will select between PC and register data A for first operand input to ALU. The fifth multiplexer will select between sign extension and register from data B. After those multiplexers have been established the control signals can be established that can be seen below in figure 2.

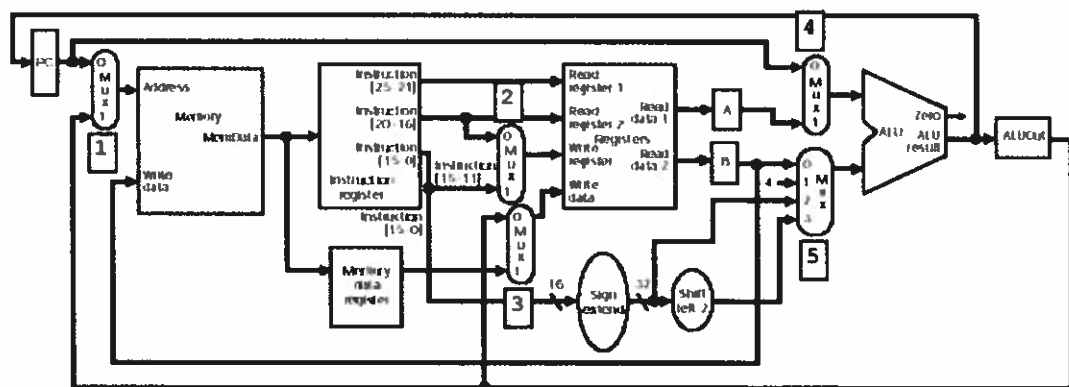


Figure 1 displays multi-cycle implementation multiplexers

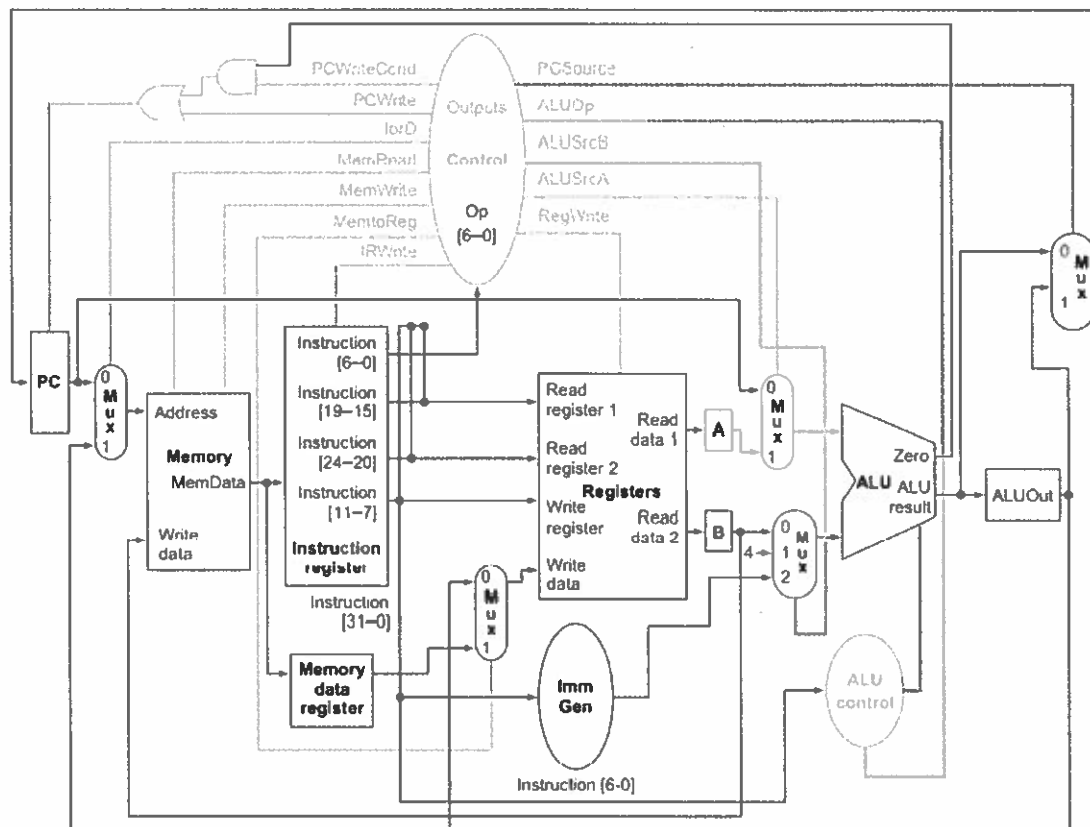


Figure 2 displays the complete data path for multicycle implementation

Multicycle implementation will be divided usually into 5 shorter cycles but will depend on the instruction and are as follows:

- Instruction fetch, program counter increment
- Partial instruction decode and branch/jump target execution
- Source operand fetch, ALU Operation, program counter update for branches/jumps
- Memory access
- Register write

Despite being significantly more efficient than single-cycle implementation, pipeline implementation is the most efficient mode that will be discussed in the following section.

## Pipelining Implementation

Pipelining requires a multicycle model. In the pipeline model as soon as a piece of hardware is finished being used after the completion of a clock cycle it is immediately used to start performing another instruction. Below is an example of a pipeline implementation from the PowerPoint notes. ✓

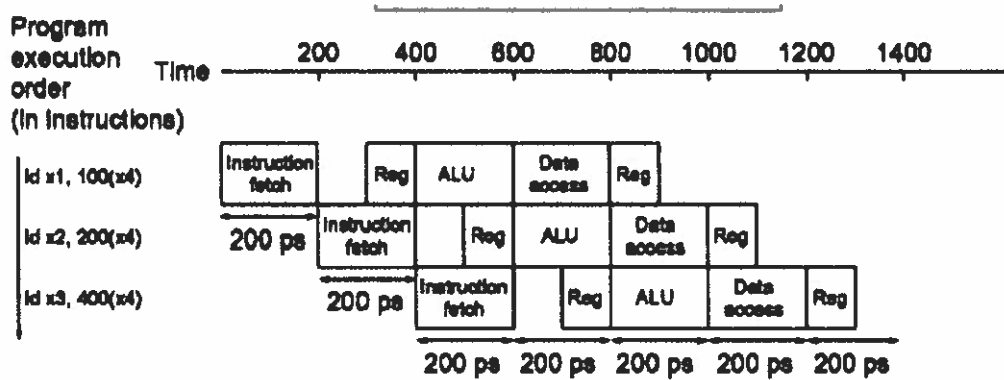


Figure 3 Pipeline Implementation

As can be seen above, for example once the first instruction is finished with its instruction fetch stage it then frees up the hardware for instruction two to then fetch during the next clock cycle.

Pipelining implementation is a technique that exploits parallelism between the instructions in a sequential instruction stream. This technique is crucial in order to increase the overall performance of a CPU. For example, pipelining helps increase the instruction throughput, and allows CPUs to work at a higher clock frequency than RAM. An increase in the number of pipeline stages increases the number of instructions executed simultaneously.

## Pipeline Hazards

When using pipelining several types of hazards can appear when the next instruction is not able to execute in the following clock cycle. There are three different types of hazards that will be explained below.

Structural hazard is when the hardware cannot support the union of instructions that are executed in the same clock cycle. This leads to the instruction to be executed in series than in parallel. The solution would be to just stall or have a pipeline bubble for that cycle.

Data hazards occur when the pipeline is stalled due to having to wait on one step and must wait for the other step to be completed. This is due to the dependence of one instruction on an earlier instruction that is still in pipeline. To solve this hazard, you can add extra hardware to retrieve the missing instruction called forwarding or bypassing. Another solution is to reorder the code to avoid use of load result in following instructions.

Control hazards are when deciding on a control action will depend on the results of one instruction while others are executing. You can insert a pipeline bubble to increase delay or use branch prediction to make educated guesses on which instruction to insert.



### Part 3: Explore the topic of modern processor design

#### Components:

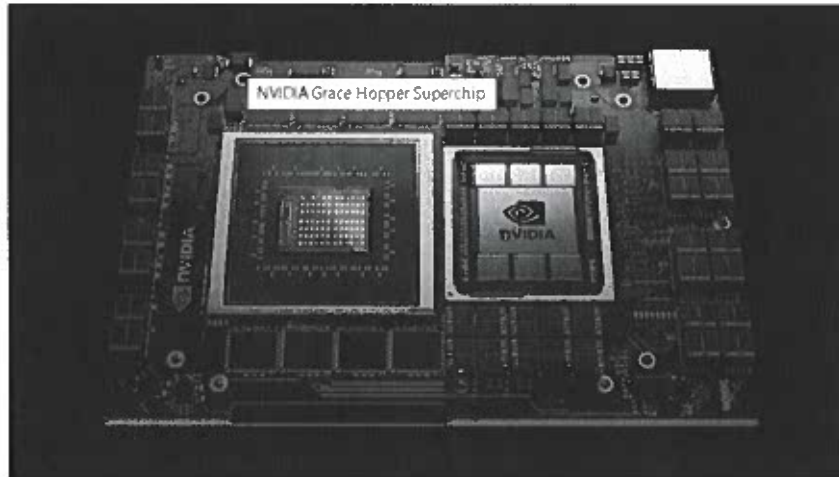


Figure 4 Grace Hopper Superchip

The modern processor explored was the NVIDIA "Grace Hopper" Superchip that plans to be released in early 2023. The processor combines a Grace architecture CPU and Hopper architecture GPU, with a NVIDIA NVLink C2C to deliver a combination of CPU + GPU memory model. This superchip is targeted for Giant artificial intelligence and high-performance workloads. ✓

The Hopper (H100) Architecture GPU will have fourth generation tensor cores that are up to six times faster than the previous A100 model. ✓

It will also include the new Tensor Memory Accelerator unit, that will assist in transferring large blocks of data efficiently between the global memory and the shared memory.

The fourth generation NVIDIA NVLink is a multi-lane near range communication link. This communication link will increase bandwidth to 900 GB/sec between the GPU and CPU and decrease latency since there is now a link between them.

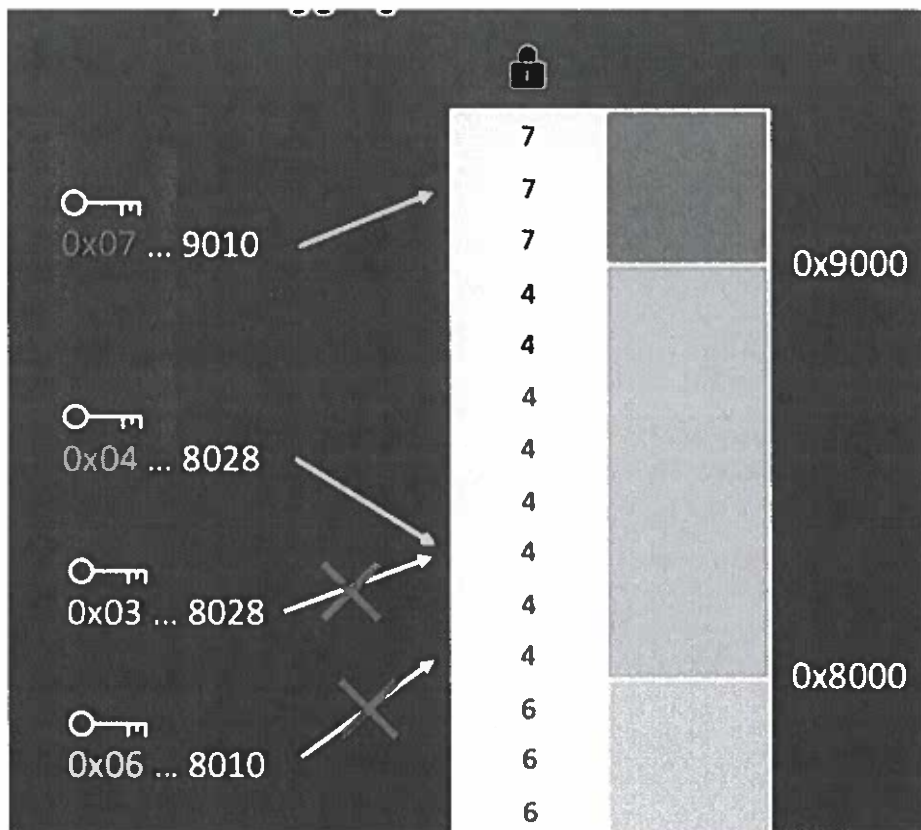
The Grace CPU will be explained in the section below.

The last feature part of the Grace Hopper Superchip is the LPDDR5X memory subsystem, a DRAM designed component to greatly boost speed, make the processor energy efficient and increasing bandwidth as well.

## How to increase ILP

Instruction level parallelism can be increased in a few ways including pipelining, multiple cores, SIMD technology, branch prediction, and much more. NVIDIA's "Grace CPU" Superchip uses 144 ARM V9 cores. ARM CPUs typically use pipelining. With 144 cores NVIDIA is using multiple cores. NVIDIA has a history of making/designing graphics cards in which they use SIMD technology, so even though not explicitly stated they are most likely going to be using SIMD in their CPUs. Another way of improving ILP is with tier 3 cache slicing. What NVIDIA has done with cache slicing is create a shared tiered 3 cache then sliced it up into smaller sections so that each slice can work independently and in parallel with each other this allows each core to access closer slices faster and if work is being performed by cores located physically closer to one core those two cores can share data faster through the tier 3 cache, which leads to less delays in the pipeline.

One of the issues that ARM tackles with the ARM V9 architecture is trying to prevent the memory access attacks that have been plaguing the CPU world. They accomplish this by using previously unused bits in the memory address pointer as a "key". Then the memory that is allocated for a program to execute is assigned a "lock" that matches the "key" in the pointer. ARM gives the figure below as an explanation.



very new processor.

limited resources

could review other related products.

**References:**

<https://www.nvidia.com/en-us/data-center/grace-cpu/>

<https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/>

<https://nvidianews.nvidia.com/news/nvidia-announces-cpu-for-giant-ai-and-high-performance-computing-workloads>

[https://www.cs.umd.edu/~meesh/cmsc311/clin-cmsc311/Lectures/lecture33/multi\\_cycle.pdf](https://www.cs.umd.edu/~meesh/cmsc311/clin-cmsc311/Lectures/lecture33/multi_cycle.pdf)

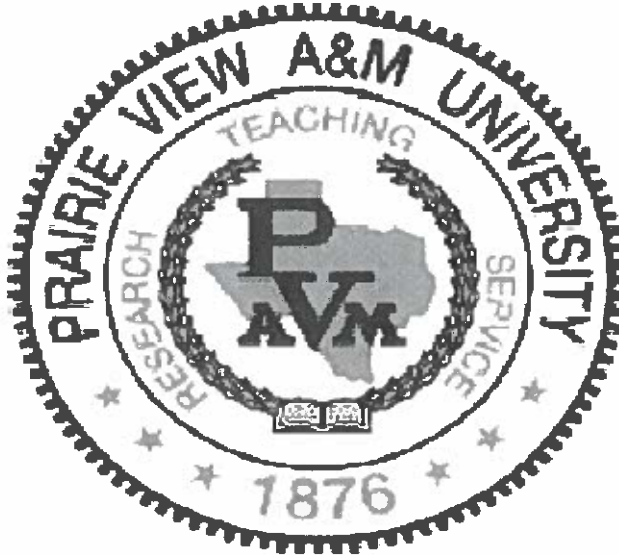
<https://wccfttech.com/nvidia-unleashes-grace-hopper-grace-cpu-superchips-platform-144-cores-up-to-600-gb-memory-2x-perf-watt/>

**Computer Organization and Design RISC-V Edition: The Hardware Software Interface**

**BELOW AVERAGE**

**Below Average Student Original Graded Assignment**

**PRAIRIE VIEW A&M UNIVERSITY**  
**ROY G. PERRY COLLEGE OF ENGINEERING**



**Course Number: ELEG 4339 – Computer Organization and Design**

**Project 2: Modern Processor Design**

**Submitted by:**

Trevor Johnson

Ryan Johnson

Alfredo Hurtado

**Date:** 4/28/2022

**Instructor:** Dr. Li

75

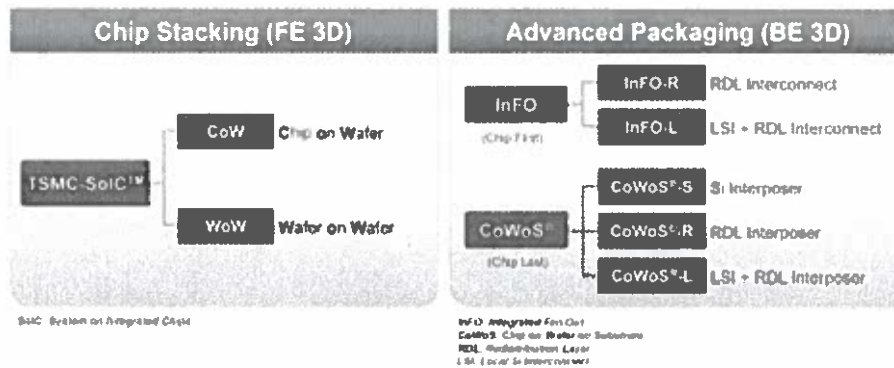
## Part 1: Discussion of Learning Strategies

The learning style I would say I am most compatible with would be visual learning style. As indicated by learning styles scholars, visual students need to see pictures. Visual students need a lot of pictures, realistic coordinators, maps, etc. to assist them with getting a handle on new data. You could advance outwardly assuming you float towards infographics and infographic recordings to learn data instead of talks or books. I perform easily when things are more hands on, and things are identified by pictures which helps my memory when performing test.

other strategies?  
more details?

## Part 2: Principles and Techniques Used in Implementing a Processor

Apple's M1 Ultra processor used TSMC's CoWoS-S (chip-on-wafer-on-substrate with silicon interposer) 2.5D interposer-based packaging process to build the M1 Ultra. Similar technologies are used by companies like AMD, Nvidia, and Fujitsu to build their high-performance processors for datacenters and high-performance computing (HPC).



Apple today announced M1 Ultra, the next giant leap for Apple silicon and the Mac. Featuring UltraFusion — Apple's innovative packaging architecture that interconnects the die of two M1 Max chips to create a system on a chip (SoC) with unprecedented levels of performance and capabilities — M1 Ultra delivers breathtaking computing power to the new Mac Studio while maintaining industry-leading performance per watt. The new SoC consists of 114 billion transistors, the most ever in a personal computer chip. M1 Ultra can be configured with up to 128GB of high-bandwidth, low-latency unified memory that can be accessed by the 20-core CPU, 64-core GPU, and 32-core Neural Engine, providing astonishing performance for developers compiling code, artists working in huge 3D environments that were previously impossible to render, and video professionals who can transcode video to ProRes up to 5.6x faster than with a 28-core Mac Pro with Afterburner.<sup>1</sup>



### Part 3: Case Study - Apple M1, M1 Ultra

Before we get into all the technical details, here are some definitions of some of the terms used in this report:

- System on a chip (SoC) - an integrated circuit (IC) that takes a single platform and integrates an entire electronic or computer system onto it.
- Central Processing Unit (CPU) - the brain of a computer, containing all the circuitry needed to process input, store data, and output results. The CPU is constantly following instructions of computer programs that tell it which data to process and how to process it.
- General Processing Unit (GPU) - a specialized processor originally designed to accelerate graphics rendering. GPUs can process many pieces of data simultaneously, making them useful for machine learning, video editing, and gaming applications.
- Advanced RISC Machines (ARM) - one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM). ARM makes 32-bit and 64-bit RISC multi-core processors.
- Efficiency core/Performance core – A performance core is a CPU core that is designed for maximum performance. Contrarily, an efficiency core delivers less raw performance but is less power hungry to conserve battery. The performance difference is obtained by varying voltages and clock speeds.

The Apple M1 is an ARM-based system on a chip designed by Apple Inc. as a CPU and GPU for its Macintosh desktops, MacBook notebooks, iPad Pro and iPad Air tablets. The M1 was released in November 2020, followed the next year by the Apple M1 Pro and M1 Max versions, and in March 2022, the latest version of the M1 generation, M1 Ultra.

The M1 is the first Apple-designed SoC that's been developed for use in Macs. It marks Apple's transition away from the Intel chips that the Cupertino company has been using in Macs since 2006. As a SoC, the M1 integrates several different components such as the CPU, GPU, unified memory architecture (RAM), Neural Engine, Secure Enclave, SSD controller, image signal processor, and more, all of which power the different features in the Mac.

Built into the M1 chip, the unified memory architecture lets the CPU, GPU, and other processor components not need to copy data between one another and are able to access the same data pool bringing notable speed and efficiency improvements to the M1. This memory architecture means that the RAM is not user upgradeable, which is expected since few Macs have user accessible RAM.

The M1 chip includes an 8-core CPU with four high-performance cores and four high-efficiency cores. The high-performance cores are designed to offer the best performance for power-intensive single-threaded tasks. The four high-performance cores can work together to offer impressive multithreaded performance that has allowed the M1 Macs to outshine even the highest-end 16-inch MacBook Pro models.

The M1 chip brings up to 3.5x faster CPU performance, up to 6x faster GPU performance, and up to 15x faster machine learning capabilities compared to the Intel chips used in prior-

generation machines. Compared to the latest PC laptop chips, the M1 offers 2x faster CPU performance and does so using just 25 percent of the power.

In the M1 family of processors, the M1 Ultra is the apex predator. Not limited to merely expand the original M1 with additional processing or graphics cores, it doubles the power by combining two M1 Max chips onto a single silicon die, with a special interconnect called UltraFusion. This high-bandwidth interconnect boasts 2.5 terabytes of bandwidth—four times that of the next competing interconnect—allowing the bifurcated design to function as a single processor on the software side.

And doubling the M1 Max means doubling pretty much everything about the chip. It's now a 20-core CPU with 16 high-performance cores and four high-efficiency cores. Graphics are handled by a powerful 64-core GPU, and memory bandwidth is expanded to a whopping 800GBps. Additionally, double the chip means double the RAM, with up to 128GB of unified memory.

*be more systematic!*

## References:

- <https://www.macrumors.com/guide/m1/>
- <https://www.pcmag.com/news/apple-m1-max-vs-m1-ultra-tested-apples-top-desktop-chips-face-off>
- <https://www.zdnet.com/article/apple-m1-ultra-blows-away-the-competition/>
- <https://www.tomshardware.com/news/apple-uses-cowos-s-to-build-m1-ultra>
- <https://www.pcmag.com/encyclopedia/term/performance-core>
- <https://www.techtarget.com/whatis/definition/ARM-processor>
- <https://www.intel.com/content/www/us/en/products/docs/processors/what-is-a-gpu.html>
- <https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:computers/xcae6f4a7ff015e7d:computer-components/a/central-processing-unit-cpu>
- <https://anysilicon.com/what-is-a-system-on-chip-soc/>

## **Students' Project Presentation**

A “mini-conference” was organized in class on Apr. 28<sup>th</sup>. Each team was given 15-20 minutes to present their findings, then followed by Q&A to the whole class. Sample presentations slides are attached here.

In general, the students did pretty good job, it demonstrates that they are interested in such topic. The topic related the trend in industry to what they have learned in the class.

Based on the presentation and the response about the questions from the class, the presentation is graded. The “above average” presentation shows clear understanding of the technical details, responded well to the questions. The “average” presentation shows a good understanding of the topic and responded ok to the questions. The “below average” presentation need more clarification during their presentation and responses.

**ABOVE AVERAGE**

**Above Average Student Original Graded Assignment**



# Project 2: Modern Processor Design

## Apple M1, M1 Ultra

Trevor Johnson

Ryan Johnson

Alfredo Hurtado



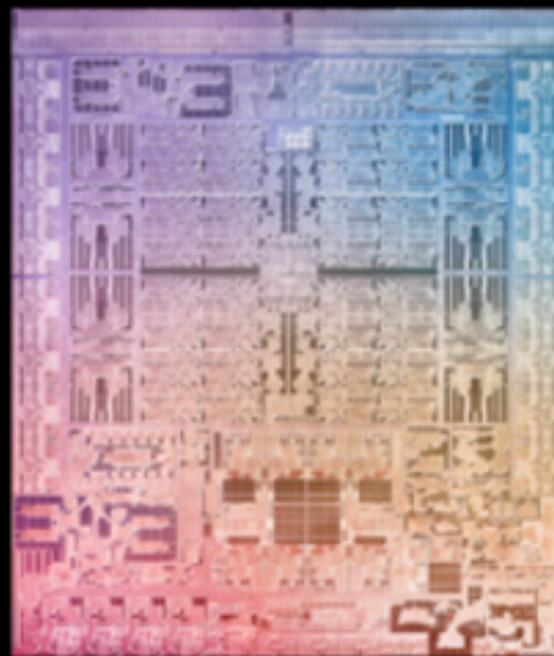
# Evolution of M1



Apple M1



Apple M1 Pro



Apple M1 Max



Apple M1 Ultra



# What is M1?

The Apple M1 is an ARM-based system on a chip (SoC) designed by Apple Inc. as a central processing unit (CPU) and graphics processing unit (GPU) for its Macintosh desktops, MacBook notebooks, iPad Pro and iPad Air tablets.



# M1

This critically-acclaimed M1 processor delivers:

- 16 billion transistors and a 119mm squared-die size.
  - 4 performance cores, 12MB L2 Cache.
  - 4 efficiency cores, 4MB L2 cache.
  - 8 GPU Cores.
  - 16GB DDR4x memory at 68GB/s.
- 





# DEVICES WITH M1 PROCESSOR

- 13-inch MacBook Pro
- MacBook Air
- Mac mini
- 24-inch iMac
- 12.9-inch iPad Pro
- 11-inch iPad Pro



# What is M1 Pro?

- The M1 Pro is Apple's second SoC developed for use in Macs, and it marks Apple's continued transition away from the Intel chips that have been used in Macs since 2006.
  - As a System on a Chip, the M1 Pro integrates several different components, including the CPU, GPU, unified memory architecture (RAM), Neural Engine, Secure Enclave, SSD controller, image signal processor, encode/decode engines, Thunderbolt controller with USB 4 support, and more, all of which power the different features in the Mac.
- 





# M1 Pro

The M1 Pro takes things higher with:

- 33.7 billion transistors on a 240mm squared die.
  - 8 performance cores, 24MB L2 Cache.
  - 2 efficiency cores with 4MB L2 cache.
  - 16 GPU Cores.
  - 32GB DDR5 memory at 200GB/s.
-

# DEVICES WITH M1 PRO PROCESSOR

- 14-inch MacBook Pro
- 16-inch MacBook Pro





# What is M1 Max?

- The M1 Max is equipped with 57 billion transistors, and it is built on a 5-nanometer process. It has 70 percent more transistors than the M1 Pro chip and it is also the largest chip that Apple has built to date.
- As a standalone chip, the M1 Max is the best Apple has to offer for MacBook Pro laptops and is available in top models of both the 14-inch and 16-inch versions.

32-core GPU



# M1 Max

The M1 Max provides:

57 billion transistors on a 420mm squared die.

8 performance cores, 24MB L2 Cache.

2 efficiency cores with 4MB L2 cache.

32 GPU Cores.

64GB DDR5 memory at 400GB/s.

---



# Devices With M1 Max

- 14-inch MacBook Pro
- 16-inch MacBook Pro
- Mac Studio



# What is M1 Ultra

- The M1 Ultra is Apple's fourth SoC developed for use in Macs, and it puts Apple one step closer to entirely transitioning away from the Intel chips that have been used in Macs since 2006.
- Compared to the M1, M1 Pro, and M1 Max, the M1 Ultra is special because it is a new feat of engineering from Apple. The M1 Ultra is essentially two M1 Max chips that have been connected through an Apple-developed method, so it offers double the performance of the M1 Max.
- The M1 Ultra features 114 billion transistors, the most ever in a personal computer chip, and it is Apple's largest Apple silicon chip to date.



# M1 Ultra

The M1 Ultra offers:

- 114 billion transistors on an 840mm (about 2.76 ft) squared die.
  - 16 performance cores, 48MB L2 Cache.
  - 4 efficiency cores with 4MB L2 cache.
  - 64 GPU Cores.
  - Up to 128GB DDR5 memory at 800GB/s.
- 



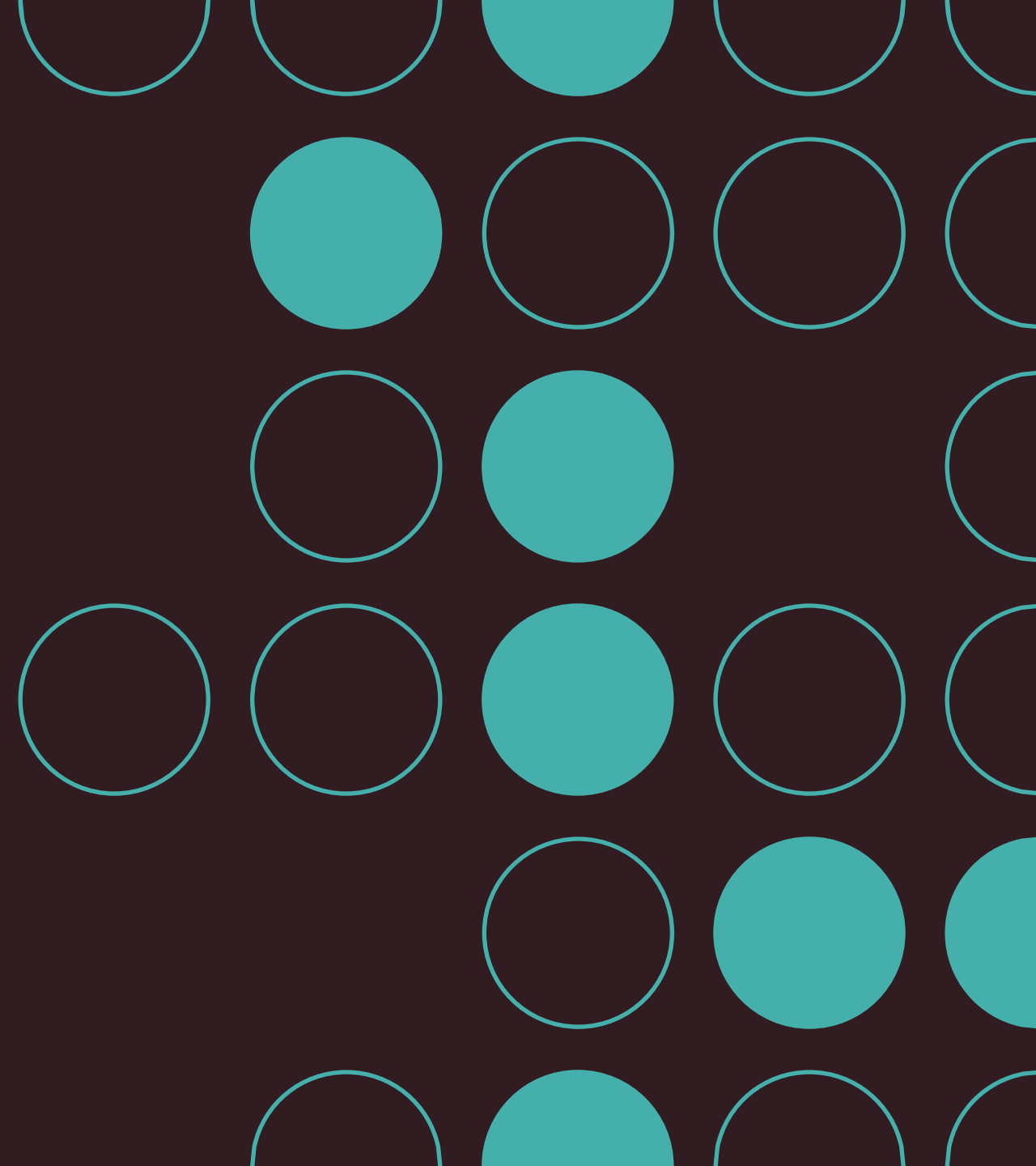


# Devices With M1 Ultra Processor

- The M1 Ultra is used exclusively in the high-end Mac Studio, which is also offered with an M1 Max chip.
-

# References

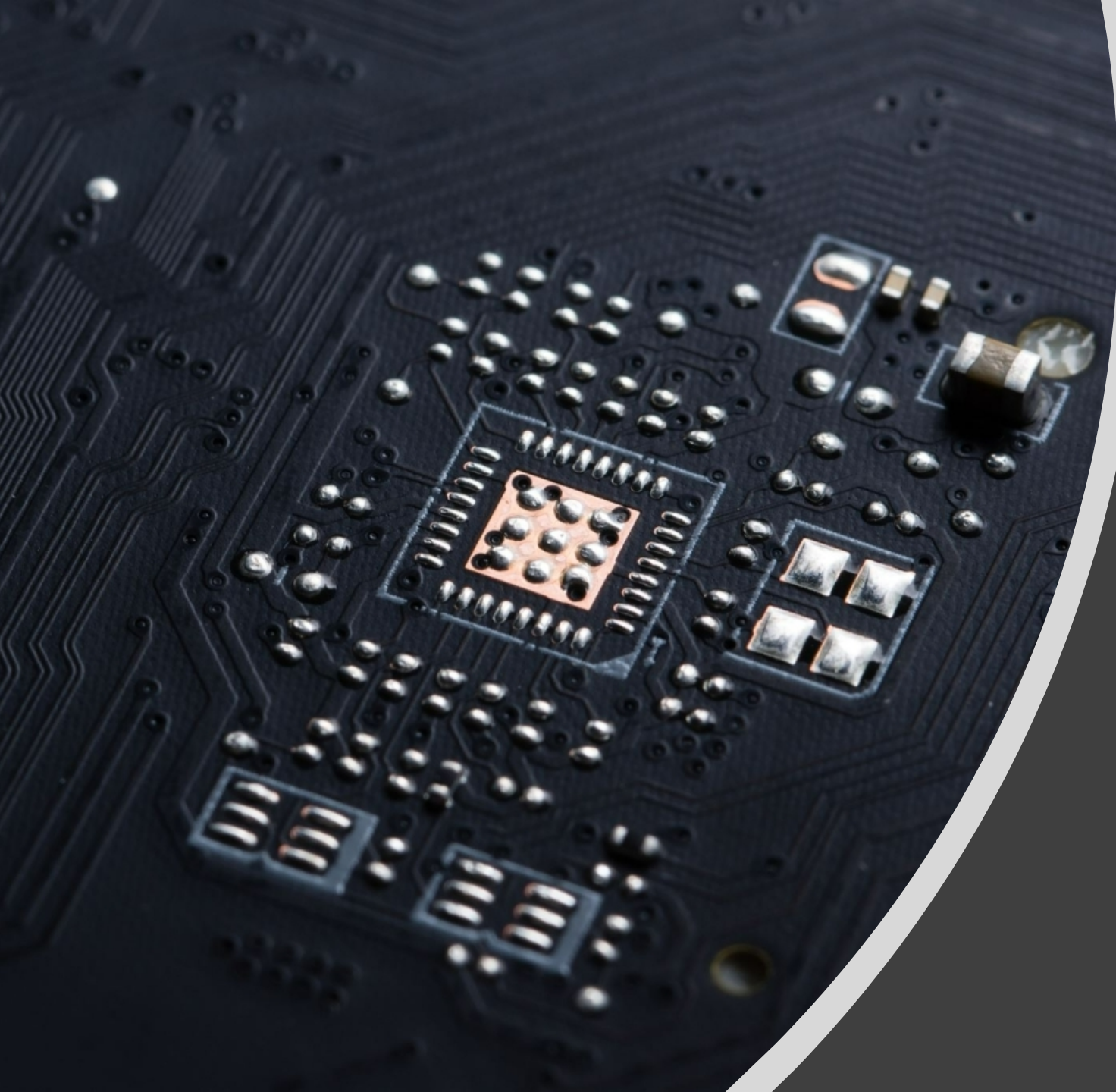
- <https://www.macrumors.com/guide/m1-pro/>
  - <https://www.macrumors.com/guide/m1-max/>
  - <https://www.macrumors.com/guide/m1-ultra/>
- 



**AVERAGE**

## **Average Student Original Graded Assignment**





# Intel Core i7

By: Johnn Owens

Terrell Gray

Nasir Pate



# Introduction

---

Intel Core i7 is a line of Intel CPUs which span eight generations of Intel chipsets. They feature either four or six cores, with stock frequencies between 2.6 and 3.7 GHz.

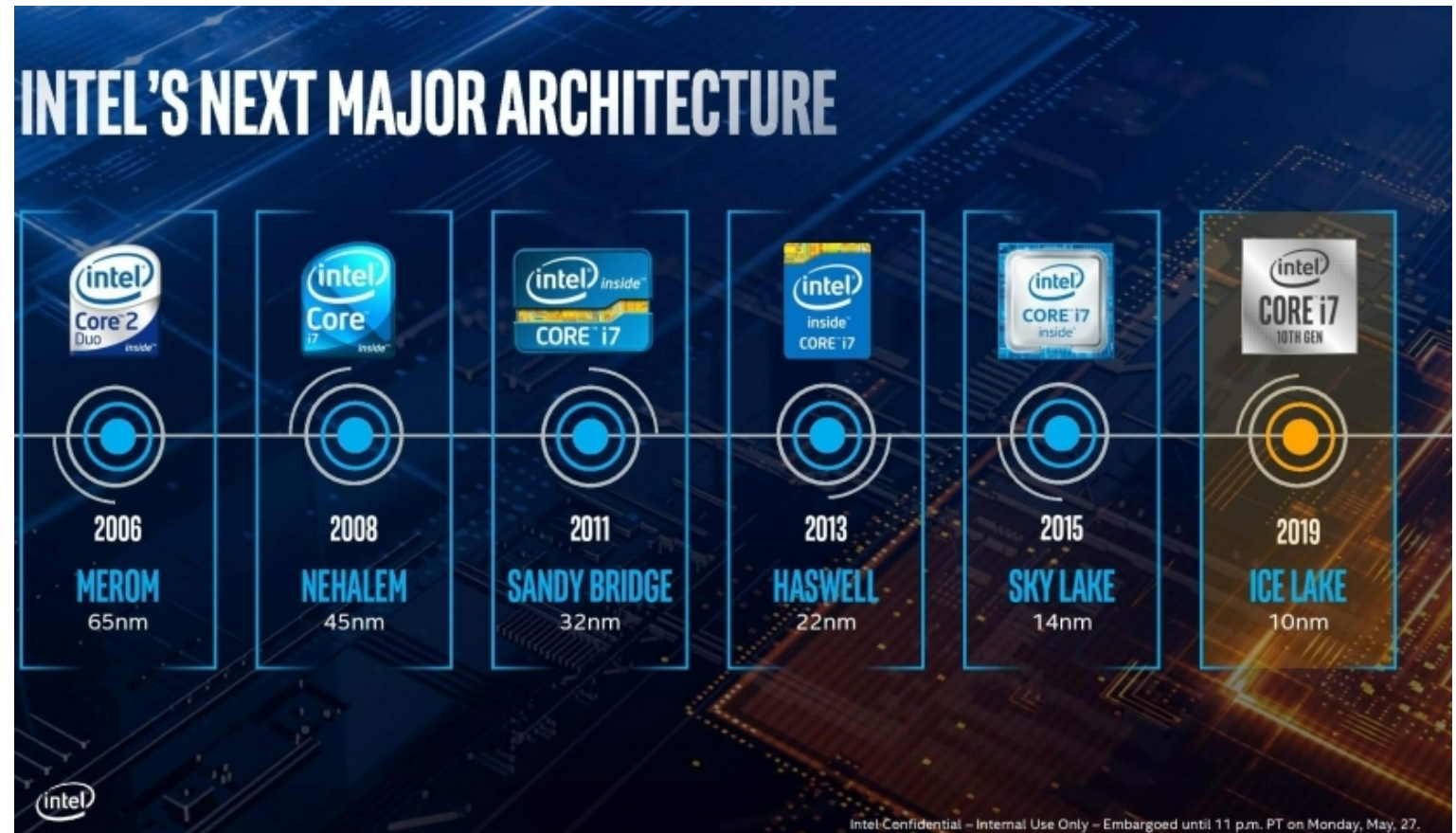
---

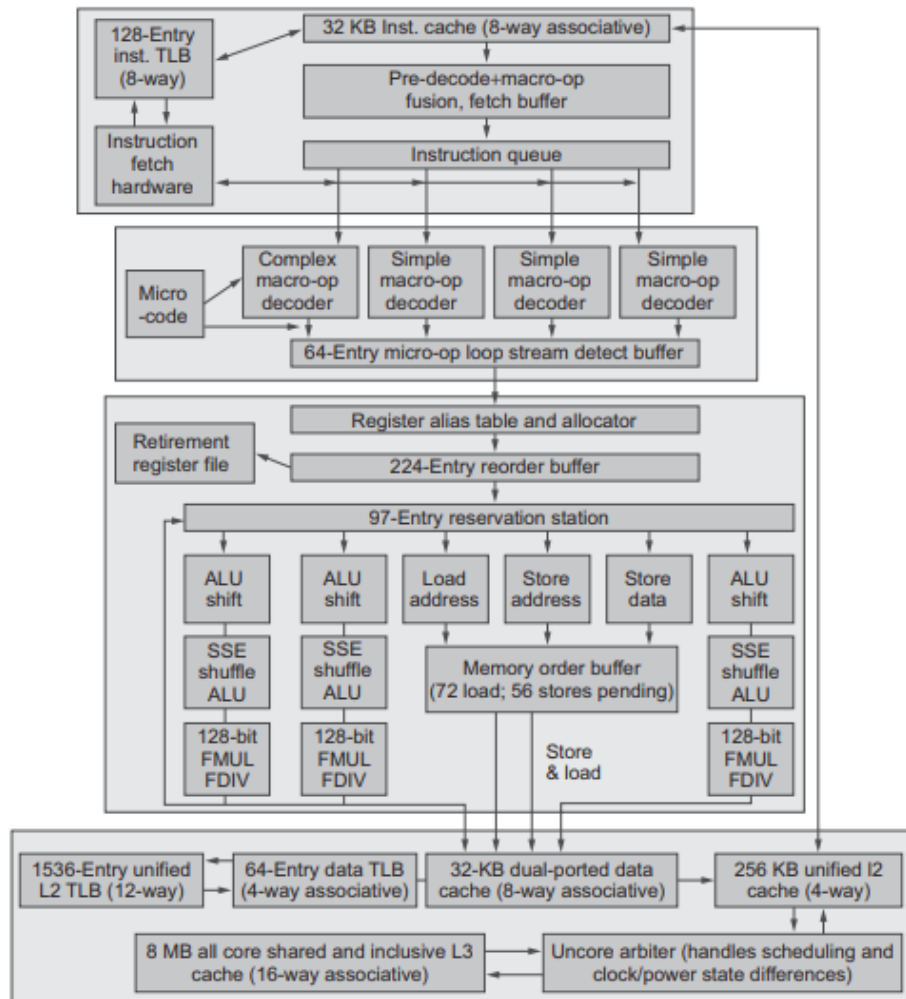


# Intel Core i7 History

The first i7 processor was introduced in 2008; the i7 is the sixth generation. The basic structure of the i7 is similar, but newer generations have enhanced performance by changing cache, increasing memory bandwidth, and expanding the number of instructions in flight.

# Intel Core i7 Comparison





**Figure 4.72: Memory System Components**

## Intel Core i7 Pipeline Structure

The total pipeline depth is 14 stages, with branch mispredictions typically costing 17 cycles. This design can buffer 72 loads and 56 stores. The six independent functional units can each begin execution of a ready micro-operation in the same cycle. Up to four micro-operations can be processed in the register-renaming table.

# Is the Intel I7 RISC or CISC?

- X86 is a family of complex set computer(CISC) instruction set architectures initially developed by Intel based on the 8-86 microprocessor
- X86 is used to describe a CPU instruction set with the Intel I7 and its' successors as well





What is the Intel Core  
i7 used for



# What the processor is used for

---

- The Core i7 is typically preferred to be used for multitasking, media editing, and high-end gaming
- The Intel Core i7 has 4 cores in its processor while having 6 cores is known as the sweet spot for maximum performance
  - The intel core i7 can compensate for its disadvantage of having less cores from their implantation of hyper-threading
  - Rather than jumping from core to core that has specific instruction sets stored in them the Intel Core i7 can execute multiple instruction simultaneously



# The Intel Core i7 Advantages

- The processor uses Hyper-Threading
- Allows more than one thread to run on each core.

More threads means more work can be done in parallel

- Hyper-Threading Technology improves CPU throughput (by up to 30% in server applications)



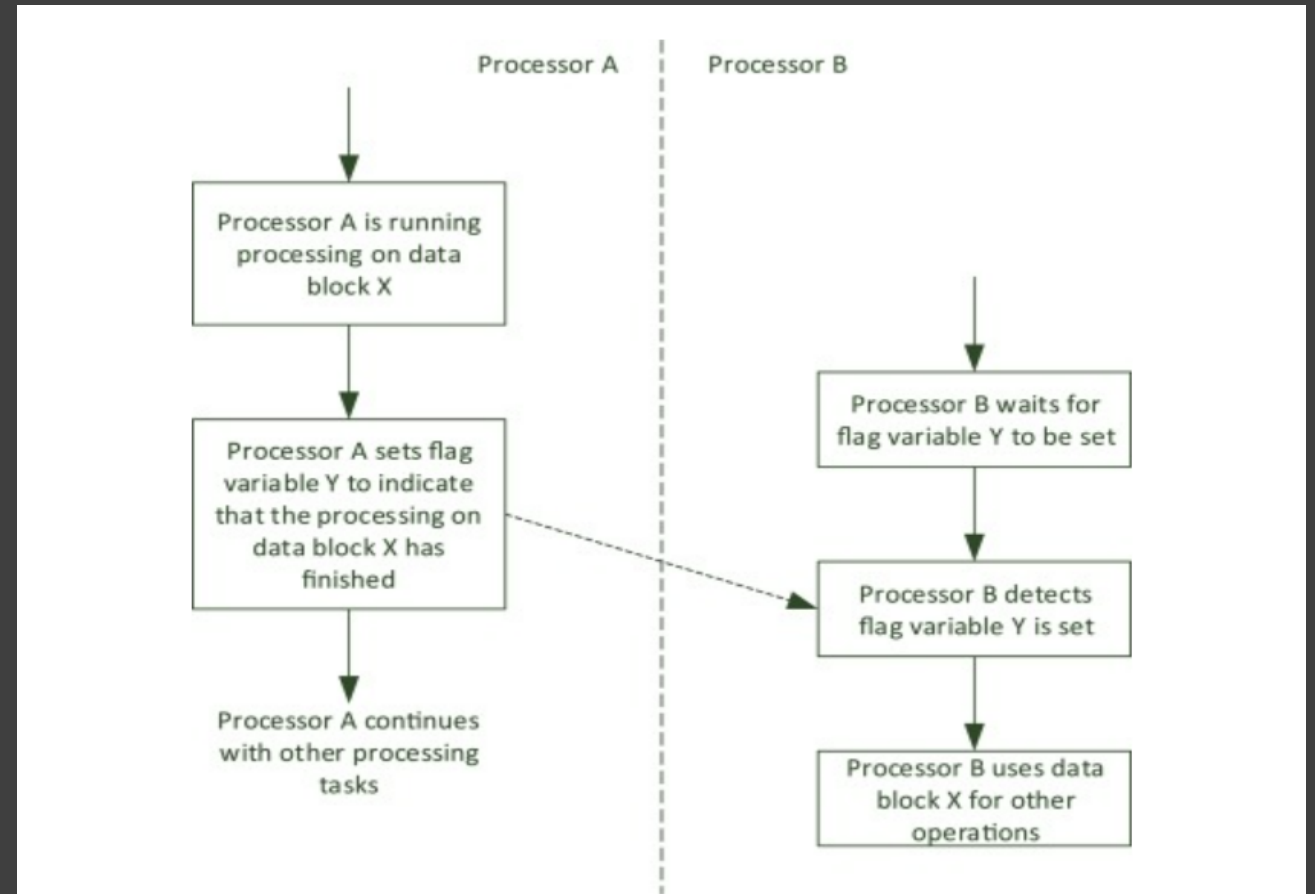
# Intel Core I7 Disadvantages

The following list includes some possible performance problems

- **Branches that are difficult to predict, causing misprediction stalls and restarts when speculation fails**
- **Long dependences—typically caused by long-running instructions or the memory hierarchy—that lead to stalls**
- **Performance delays arising in accessing memory causes the processor to stall**

# Modern Processor design to improve performance

- Memory access reordering
  - A processor can potentially perform the read operation earlier to prevent subsequent data processing from being stalled
- The memory accessing is a suitable option but when addressing two or more processors(shown in the example)
  - Processor B will take data from memory location X that has not been updated which will result in a stall



# Deeper Pipelines

- Chopping the pipeline in more stages is the easiest way to increase performance
- It can run faster because each step contains less logic, so it is able to run faster
- Consequently, Deeper pipelines can result increase the cost because of extra pipeline registers and hardware required for hazards

# The Intel Core i7 Processor



The heart of this system design is the Intel® Core 7 processor, a high-end 64-bit implementation of the Intel architecture. Including several notable features:

- Four independent CPU cores
- Two-way multithreading per CPU core
- A built-in two-channel DDR3 DRAM controller
- Integrated L1, L2, and L3 caches
- Direct Media Interface (DMI) connection between the processor and the PCH

# Performance & Comparison

- The clock rate is 3.4 GHz on the i7 versus up to 1.3 GHz for the A53
- The average instruction time for the Intel i7 is  $0.64 \times 1/3.4 \text{ GHz} = 0.18 \text{ ns}$
- While the average instruction time for the A53 is  $1.36 \times 1/1.3 \text{ GHz} = 1.05 \text{ ns}$
- This means that the I7 moves 5x as fast, but also uses 200x the power
- High energy processor which could lead to possible overheating

# Sources

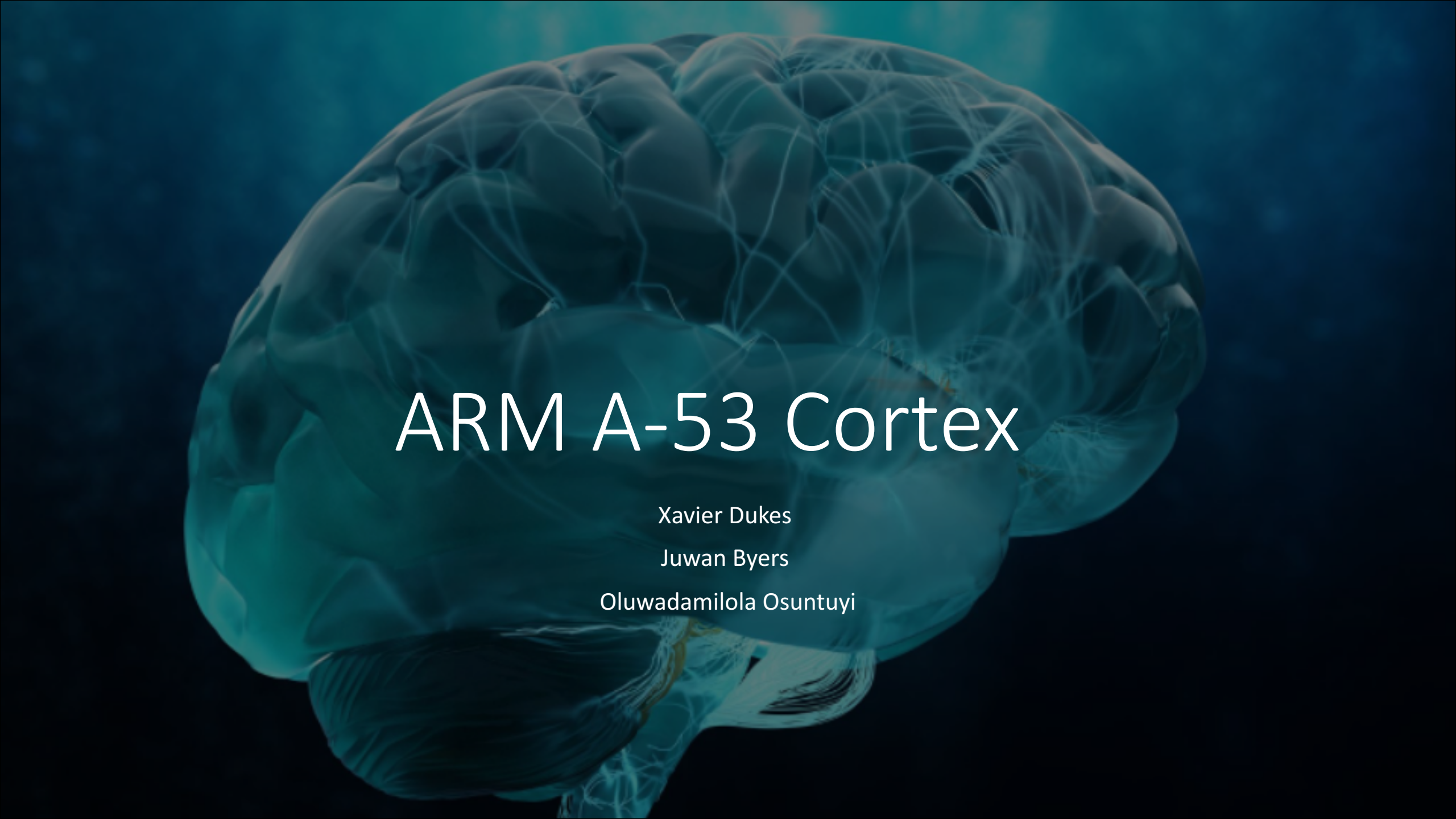
- “Intel Core i7-12700K Benchmarks.” Intel Core i7-12700K Benchmarks - Geekbench Browser, <https://browser.geekbench.com/processors/intel-core-i7-12700k>.
- Patterson, David A., and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann Publishers, an Imprint of Elsevier, 2021.
- Here’s how Intel® Hyper-Threading Technology (Intel® HT Technology) helps processors do more work at the same time.[d:604] Here’s how Intel® Hyper-Threading Technology (Intel® HT Technology) helps process, Intel. “What Is Hyper-Threading?” Intel, 2021, <https://www.intel.com/content/www/us/en/gaming/resources/hyper-threading.html>.
- Ibrahim, Dogan. “Processor Performance.” *Processor Performance - an Overview | ScienceDirect Topics*, 2021, <https://www.sciencedirect.com/topics/engineering/processor-performance>.

ANY  
QUESTIONS  
?

**BELOW AVERAGE**



**Below Average Student Original Graded Assignment**

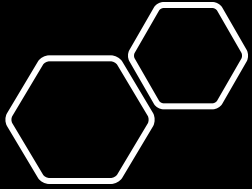


# ARM A-53 Cortex

Xavier Dukes

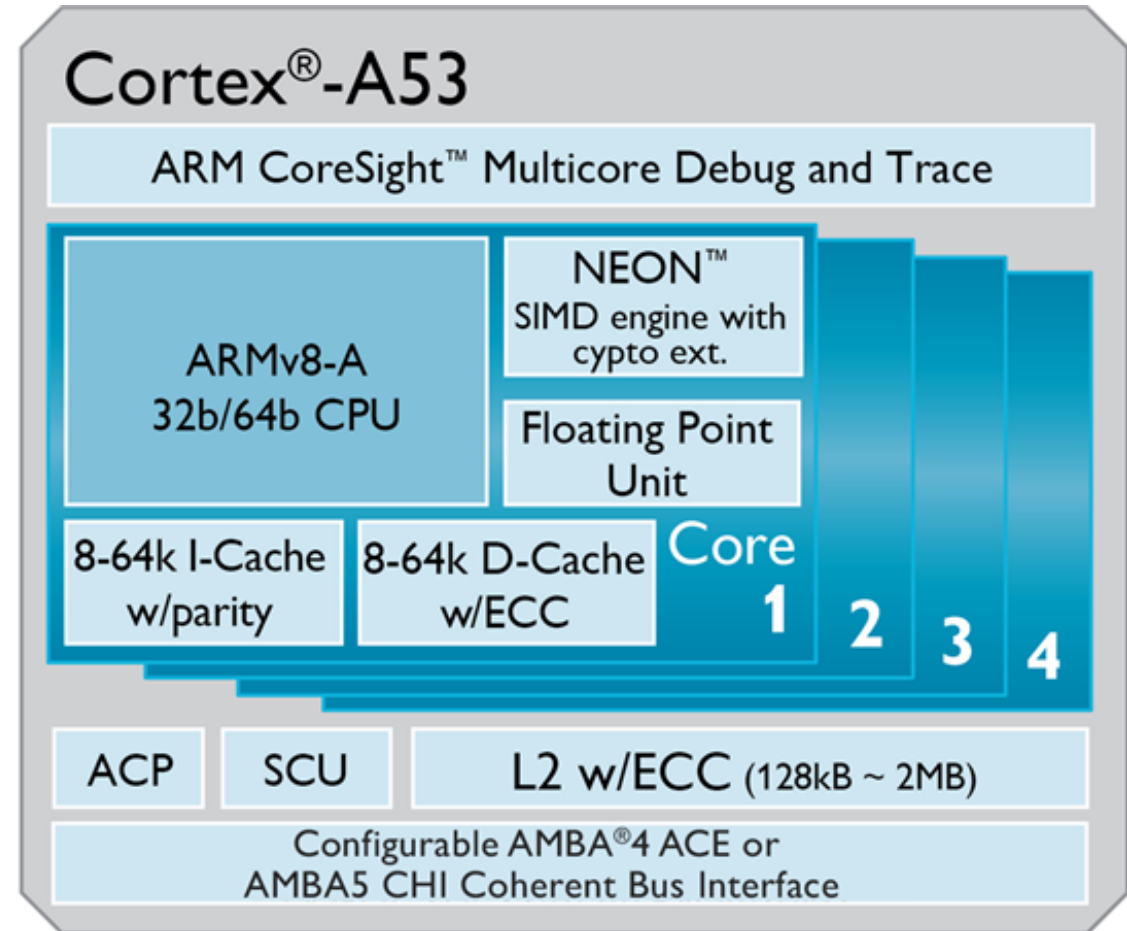
Juwan Byers

Oluwadamilola Osuntuyi

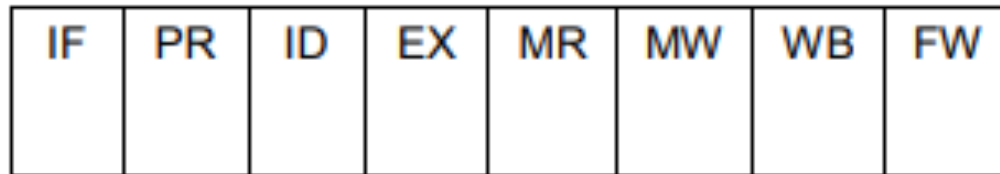


# ARM Cortex

- 1-4 core, 64 bit, dual issue pipelined model of processor
- CPU and Floating point Processing Unit to handle various types of arithmetic
- 8- stage execution of instructions with a max clock rate of 2.65 GHZ
- common in mobile devices and embedded systems such as the Raspberry Pi 3 and Zero



# Breakdown of 8-stage pipeline



**Figure 1: Pipeline Stages**

- IF: instruction fetch
- PR: predict and register
  - Branch predict
  - Jump-target predict
  - Register file read/write
- ID: decode
  - Mult/Div
- EX: execute
  - Resolve branches
- MR: memory read
- MW: memory write
- WB: write back
- FW: forward
  - forward WB value to ID stage

# Instruction Set (ARMv8)

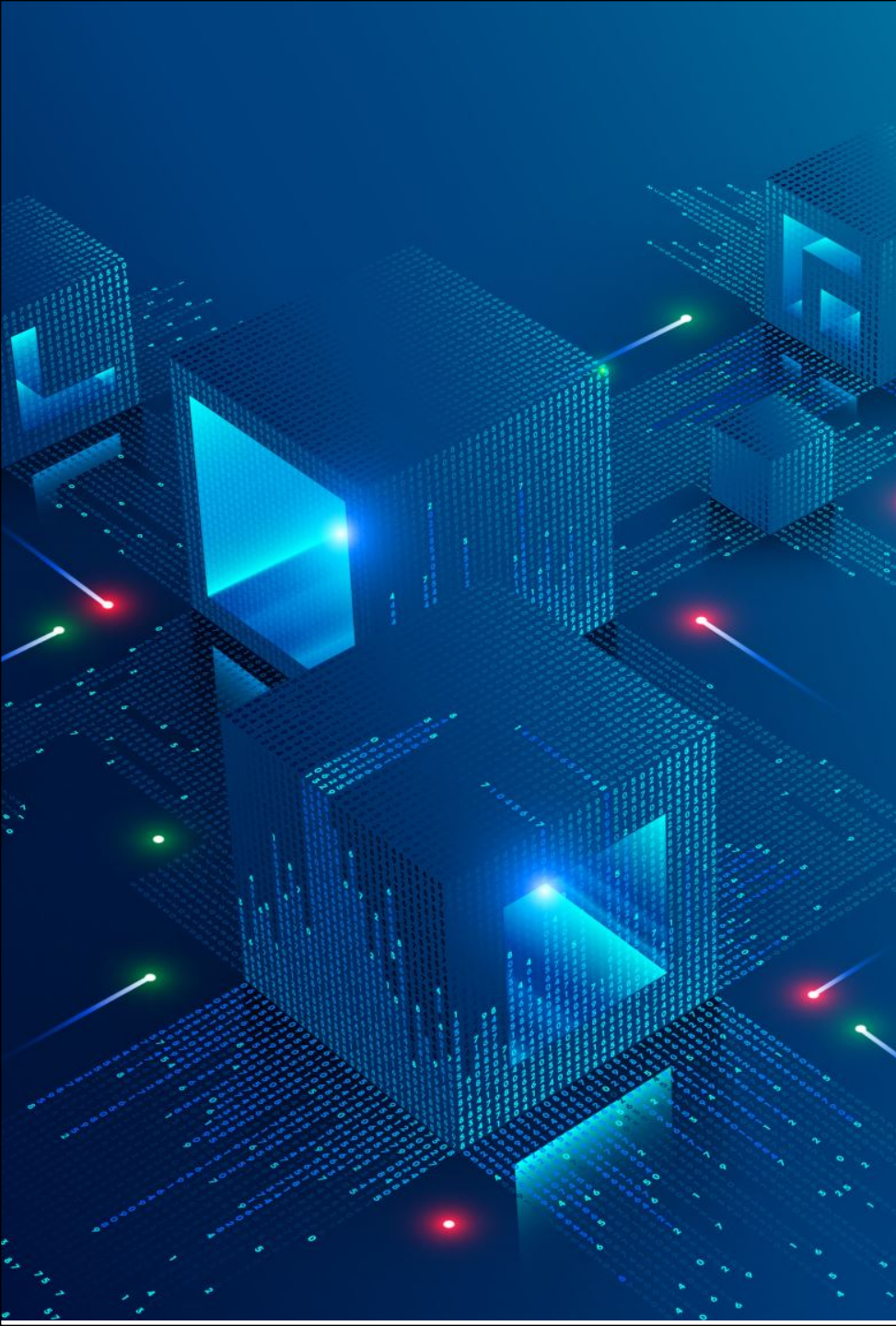
Adds an optional 64-bit architecture, AArch64, and A64 instruction set.

AArch64 provides user-space compatibility with ARMv7-A

Thumb instruction set has no 64-bit counterpart

Allows 32-bit applications to be executed in a 64-bit OS, and a 32-bit OS to be under the control of a 64-bit hypervisor

Used with Apple A7 in Iphone 5S and Exynos 5433 in Galaxy Note 4



# Increase of cycle per instruction

---

- Utilizes the Harvard structure
  - computer architecture with separate storage and signal pathways for instructions and data, this helps ensure that structural hazards do not occur and therefore encroaches on the ideal CPI
  - allows for some redundancy and the freeing of various resources and therefore avoiding possible structural hazards
    - buses, L1 & L2 Data and instruction Caches for each core
    - L1 cache is 8 to 64 KiB and L2 cache is 128KiB to 2 MiB
- Specialized hardware to handle encryption or floating values via FPU



# Control Hazard and Preloading

- Control hazard techniques
  - The processor makes use of branch prediction and speculative instruction fetching leading to a pipeline flush depending on if the branch condition is true or false
    - pipeline flush will dump the fetched instruction and begin execution of the proper instruction
- Preloading instructions and Data prefetching
  - The data cache implements an automatic prefetcher that monitors cache misses in the core. When a pattern is detected, the automatic prefetcher starts linefills in the background. The prefetcher recognizes a sequence of data cache misses at a fixed stride pattern
  - Instruction Fetching Unit that sequentially grabs instructions from a 2 way cache and “ obtains instructions .... and predicts the outcome of branches in the instruction stream

# Future Generations vs Older Generation

| ARM Generations  | Instruction Set | L1, L2, L3 Cache                                                    | Cores                                                       | Clock Rate                                             |
|------------------|-----------------|---------------------------------------------------------------------|-------------------------------------------------------------|--------------------------------------------------------|
| ARM Cortex A-53  | ARMv8           | L1 = 8 to 64 KiB<br>L2 = 128 KiB to 2 MiB                           | 1-4 per cluster                                             | 957 MHz to 2.65 GHz                                    |
| ARM Cortex A-55  | ARMv8.2         | L1 = 32 to 128 KB<br>L2 = 64 to 256 KB<br>L3 = 512 KB to 4 MB       | 1-8 per cluster, multiple clusters                          | 1.8GHz to 2.3 GHz                                      |
| ARM Cortex A-510 | ARMv9           | L1 = 32 KB to 64 KB<br>L2 = 128KB to 512 KB<br>L3 = 256 KB to 16 MB | Up to 8 cores per cluster, Merged-core, 2 cores per complex | CPU not listed but Hypertransport is a speed component |



# References

Developer, A. R. M. (n.d.). Documentation – arm developer. Retrieved April 27, 2022, from <https://developer.arm.com/documentation/ddi0500/j/Level-1-Memory-System/Data-prefetching/Data-prefetching-and-monitoring?lang=en>

Developer, A. R. M. (n.d.). Cortex-A53. Retrieved April 27, 2022, from <https://developer.arm.com/Processors/Cortex-A53>

Ltd., A. (n.d.). Cortex-A53. Arm. Retrieved April 27, 2022, from <https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a53>

Wikimedia Foundation. (2022, April 6). ARM cortex-A53. Wikipedia. Retrieved April 28, 2022, from [https://en.wikipedia.org/wiki/ARM\\_Cortex-A53](https://en.wikipedia.org/wiki/ARM_Cortex-A53)

