HW3: Processor

Due November 20, 2020, 11:55pm

If you have questions, use the piazza forums (and professor/TA office hours) to obtain assistance.

Task:

There are 2 parts to this assignment: part 1: written exercises for Ch4; part 2: MIPS programming. Basic rules regarding assignment implementation and submission:

- Total points of this assignment: 100.
- **Team Allowed. Maximum 2 per Team**. You should be able to specify your team on gradescope. To be safe, include **names and G#'s of group members in ALL submitted files.**

Submission Instructions

- Submit to **gradescope**. A link to gradescope is available from Blackboard. There will be two assignments on gradescope, one for part 1 and one for part 2.
- For Part1, follow the submission instruction from gradescope. You will need to upload a PDF and specify which page we can find your answer. Make sure your answer is legible.
- For Part 2, name your MIPS source code as
  `cs465_hw3_username1[_userame2].asm`
  - Here username is the first part of your GMU email address. Make sure to include both members' names if you have two people in the team.
  - For example: cs465_hw2_yzhong.asm
  - Do **NOT** submit it as a .pdf.
- **Plagiarism is not permitted in any form. I enforce the university honor code.**

Part 1. Written Exercise for Processor (45%)

Notes:

- **A large portion of (or all) points will be taken off if you do not include detailed calculation in your answer.** You must show steps to justify your answer.
- **Answers must be legible**, especially if you scan to generate your submission.

1. (15 pts) Assume that we only consider the following latencies for elements in a datapath as in Figure 4.17 of the textbook. Answer the following questions and show your calculation to get full credit. **Note**: **Regs** and **D-Mem** latencies in the table below apply to both reading and writing of register file and data memory.

I-Mem	Regs	ALU	D-Mem	Sign-extend
400ps	200ps	150ps	450ps	60ps

1.1 (4 pts) How long does it take to complete the execution of a BEQ instruction?
1.2 (4 pts) How long does it take to complete the execution of a SUB instruction?
1.3 (4 pts) How long does it take to complete the execution of an LW instruction?
1.4 (3 pts) Suppose that we only consider BEQ, SUB, and LW instructions and use the same clock cycle time for all three. If we can reduce the latency of only one given datapath element by 25%, which element should we pick? Why?

2. (19 pts) Given the following sequence of instructions to be executed on a 5-stage pipelined datapath as descripted in our textbook:

```
I0:   add   $8, $9,$10
I1:   add   $11,$11,$8
I2:   lw    $8,0($9)
I3:   or    $8,$8,$10
I4:   sw    $11,0($8)
```

2.1 (5 pts) List true dependencies in the given sequence in the format of (register_involved. producer_instruction, consumer_instruction). Use labels to indicate instructions. For example: `($1, I10, I11)` means a true dependence between instruction `I10` and `I11`: value of register `$1` is generated by `I10` and used by `I11`. Do **NOT** list output dependence or anti-dependences.

2.2 (5 pts) If there is no forwarding and no reordering, insert nops to ensure correct execution. Show the sequence of execution with nops.

2.3 (9 pts) If there is full forwarding support, draw multiple-cycled pipeline diagram (like Figure 4.44) to show the execution of the sequence. Use arrows to mark forwardings clearly in your diagram. Each arrow should point from instruction/stage handing off the data → instruction/stage receiving the data. Also mark the necessary pipeline stalls.

3 (11 pts) This exercise examines the accuracy of various branch predictors.

3.1 (2 pts) Consider the branch sequence: **NT, T, T, NT, T, T**. What is the accuracy of always-not-taken predictor for this sequence? Show your calculation.

3.2 (6 pts) Fill in the table below to show the status transition/prediction of a two-bit predictor for the same branch sequence. Assume that the predictor starts off in the top right state from Figure 4.63 ((weak) predict taken).

Branch behavior	**NT**	**T**	**T**	**NT**	**T**	**T**
Predictor status	**Weak T**					
Prediction	**T**					
Correct prediction?	**No**					

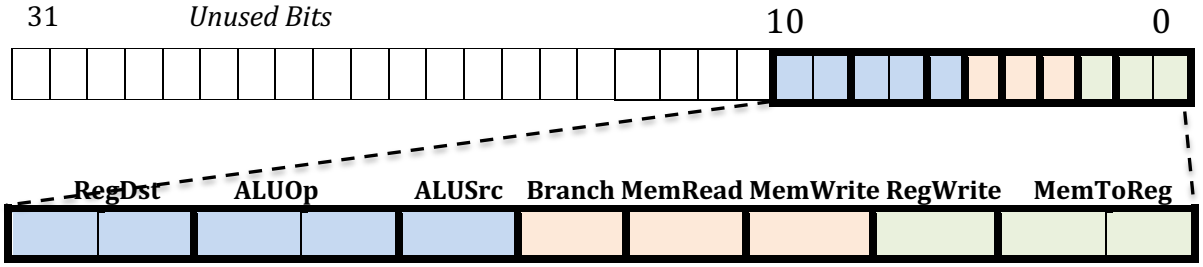
3.3 (3 pts) What is the accuracy of this two-bit predictor for the given sequence of 6 branches based on your table above? What is the accuracy if the same branch sequence repeats forever? Show your calculation.

Part 2. MIPS Programming (55%)

MIPS Pipeline Decoder. For this assignment, you will write a program to accept a sequence of machine code (as hexadecimal strings) from the user, decode, identify and report control signals and data dependences for the given sequence.

Main program: Your main program must perform the following tasks:

- Accept an integer **N** from the user which specifies how many instructions to process.
  - You can assume this is always a non-negative integer.
  - You can assume **N** is no greater than 10.
- Accept a sequence of **N** machine instruction words as a sequence of strings from the user.
  - Every instruction is given as one hexadecimal string from the user. (Same as HW1/2)
  - Please download and use the provided **cs465_hw3_template.asm** as your starting point. It already includes MIPS lines that accept **N** and a sequence of **N** instructions from the standard input. Do NOT change the provided part but you may need to add code to store the instructions/instruction features for later processing.
- Extract the numeric value from each input string and generate a label for each of them based on their order in the given sequence.
  - Every input is a valid hexadecimal value, for example **" 012A4020"** or **" AE1F0010"**.
  - You can assume only capital case letters (**'A'** to **'F'**) will be used for the input.
  - The first instruction should get label **I0**, the second instruction gets label **I1**, etc.
  - You only need to support a subset of MIPS instructions: **add, addi, and, slt, lw, sw, beq**. (We drop **j** for this assignment.)
  - You can assume all instructions from the user are valid supported instructions.
- Decode each instruction and report the control signals generated for that instruction.
  - You only need to consider the control signals in Figure 4.49 of the textbook (i.e. no controls discussed in Ch4.7 needed for this task).
  - As part of the task, decide how control signals should be set for **addi** and include a brief explanation in your comments.
  - You need to assemble all control signals together as the lowest 11 bits of a 32-bit integer following this format:



    - RegDst: bit 10-9, ALUOp: bit 8-7, ALUSrc: 6 Branch: bit 5, MemRead: bit 4, MemWrite: bit 3, RegWrite: bit 2, MemToReg: bit 1-0.
    - Encode RegDst and MemToReg: 00 for signal 0; 01 for signal 1, 11 for signal X.
    - Unused bits should be set to zero
  - You will need to print out the control signal word as 8 hexadecimal digits.
- Identify and **report** the instruction(s) that each instruction is data-dependent on.
  - Only report true data-dependence in the same format as Part 1 Question 2.1.
  - You can ignore the control flow caused by branches, which means that you only need to analyze the given instructions in sequential order.

Coding Requirements:

- **You must use the provided template to accept input from the user.**
- You must mark clear the start and end of all required steps.
- You code must be very well commented. A description of the algorithm you use must be included in your comments.
- Feel free to define helper functions but there is no required one for this homework.
- You might find the provided or your own MIPS code from HW1 and/or HW2 helpful. Feel free to use them for this assignment.

Sample run (user input in underlined blue, with newline displayed explicitly):

```
How many instructions to process? 4↵
Please input instruction sequence (one per line):
032AC020↵
030AC820↵
03195820↵
02EAC020↵
I0: Control signals: 0x00000304
Dependences: None
-----------------------------------------
I1: Control signals: 0x00000304
Dependences: (24, I0, I1)
-----------------------------------------
I2: Control signals: 0x00000304
Dependences: (24, I0, I2), (25, I1, I2)
-----------------------------------------
I3: Control signals: 0x00000304
Dependences: None
-----------------------------------------
```

```
#I0: add $24, $25, $10
#I1: add $25, $24, $10
#I2: add $11, $24, $25
#I3: add $24, $23, $10
```

```
#I0: first instruction, no dependence
Control signals = 01 10 0 0 0 0 1 00
```

```
#I1: need to use $24 updated by I0
```

```
#I2: dependences for both source registers
```

```
#I3: no true dependence
```

- **Sample run obtained from command-line executions in a prompt. Newline and user input display will be different if you run directly in MARS but all numbers/strings should match.**

Grading rubric:

• Code assembled with no error	5/55
• Code well commented	5/55
• Loop through all instructions	5/55
• Report control signals correctly	20/55
• Report dependences correctly	20/55

Appendix: MIPS Pipeline

You need to support the typical 5-stage pipelined processor and a subset of MIPS instructions. We also include some assumptions to simplify the problem of this assignment. You should get the details from our textbook/lecture slides, but this section serves as a quick reference for the main features and assumptions we make for this assignment.

Pipelined processor that you need to support:

Pipeline features:

- There are five stages of pipeline execution: IF, ID, EX, MEM, and WB
- Register reading is performed at ID stage while register writing is performed at WB stage.
- Register file can be updated in the first half of a clock cycle and read in the second half of a clock cycle.
- For branches, we always assume branch-not-taken and therefore continue for sequential transitions.

Data dependences:

- You need to identify and report true data dependences only.
  - Example 1:
    - `I0: add $t0, $t1, $t1`
    - `I1: add $t2, $t1, $t0`
    - `I2: add $t0, $t2, $t3`

    ```
    # true dependence between I0 and I1: ($t0, I0, I1)
    # true dependence between I1 and I2: ($t2, I1, I2)
    # I0 and I2 have an output dependence with $t0: no report
    # I1 and I2 have an anti-dependence with $t0: no report
    ```

  - Example 2:
    - `I0: add $t0, $t1, $t1`
    - `I1: add $t0, $t1, $t0`
    - `I2: add $t2, $t0, $t3`

  ```
  # true dependence between I0 and I1: ($t0, I0, I1)
  # true dependence between I1 and I2: ($t0, I1, I2)
  # no true dependence between I0 and I2
  ```

- We ignore control transitions caused by branches (forward or backward). You only need to consider sequential transitions for this assignment.

  - Example 1
    - `I0: add $t0, $t1, $t2`
    - `I1: add $t2, $t1, $t3`
    - `I2: beq $t3, $zero, I0`

    ```
    # for I0 - No report dependence of ($t2, I1, I0)
    # since I0->I1->I2->I0 only happens if the branch of
    # I2 is taken, which we ignore for this assignment
    ```

- Example 2
  - `I0: add $t0, $t1, $t2`
  - `I1: beq $t3, $zero, I3`
  - `I2: add $t2, $t4, $t0`

```
# for I2 – report dependence ($t0, I0, I2) since we assume
# sequential execution (branch is not taken).
```

MIPS instructions that you must support:

You are required to support only a subset of MIPS instructions. Use the table below as your quick reference. This is (almost) the same as the one for our HW 2. For this assignment, you can assume the input sequence only contains valid instructions.

MIPS	opcode	funct	Type	MIPS	opcode	funct	Type
add	0x00	0x20	R (0)	addi	0x08	X	I (1)
and	0x00	0x24	R (0)	slt	0x00	0x2a	R (0)
lw	0x23	X	I (1)	sw	0x2b	X	I (1)
beq	0x04	X	I (1)	~~j~~ (not for HW3)	~~0x02~~	~~X~~	~~J (2)~~