HW3: Processors

Due April 16th, 2023, 11:59pm

extra credit available for early submissions!

Jalcant4 G00845927 **Aren4 G01135138**

Part 1. Written Exercise for Processors (50%)

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. (10 pts) Assume that we only consider the following latencies for elements in a single-cycle processor as in **Figure 4.17** of the textbook (or Slide 49 of Lec6.processor.part3.pdf).

I-Mem	Reg reading	Reg writing	ALU	D-Mem reading	D-Mem writing	Sign-extend
350ps	150ps	150ps	150ps	400ps	400ps	80ps

The execution of an instruction typically follows multiple parallel paths to propagate information in the datapath. The **critical path** of the execution is the path that takes the longest time to complete. For example, the critical path of BEQ with the given delays above is **I-Mem□Reg reading□ALU**. The latency to complete a BEQ is therefore 350+150+150 = 650ps. Apply the same idea to answer the following questions:

1.1. What is the critical path of an ADD instruction? How long does it take to complete the execution of an ADD instruction.

- **The critical path of an ADD instruction is I-Mem-> Reg reading -> ALU -> Reg Write. If you add the latency of each element together it comes out to 350 + 150+ 150+ 150 = 800ps.**

1.2. What is the critical path of an LW instruction? How long does it take to complete the execution of an LW instruction?

- **The critical path of an LW instruction is I-Mem -> Reg reading -> ALU -> D-Mem Reading -> Reg Writing. If you add the latency of each element together it comes out to 350 + 150 + 150 + 400 + 150 = 1200ps.**

2. (8 pts) Consider the control signals defined in the single-cycle processor as in **Figure 4.17** of the textbook (or Slide 49 of Lec6.processor.part3.pdf). Suppose we now need to support an additional instruction `addi`. How should we set the control signals of `Branch`, `RegWrite`, `MemRead`, `MemWrite`, `RegDst`, `ALUSrc`, `MemtoReg`, and `ALUOp`? Include a brief explanation for each of the signals.

- **The Branch should be 0 as addi does not branch.**
- **RegWrite should be 1 since we write the result of addi into a new register.**
- **MemRead and MemWrite should be 0 as it does not write or read into memory.**
- **RegDst should be 0 as it writes the result to rt.**
- **ALUSrc should be 1 since one of the inputs is an immediate value.**
- **MemtoReg should be 0 as it does not use memory.**
- **ALUOp should be 10 because the ALUOp for add is 10.**

3. (22 pts) Given the following sequence of instructions to be executed by a 5-stage pipelined processor as described in our textbook:

```
I1:   and   $11,$2,$12
I2:   lw    $24,0($10)
I3:   addi  $24,$24,4
I4:   addi  $10,$10,4
I5:   sw    $11,0($24)
```

3.1 (6 pts) List **<u>true dependencies</u>** with registers 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`. **NOTES**:

- If a register has been updated by multiple instructions, only consider the **most recent** instruction that updated the register;
- Do not list output or anti-dependences;
- Do not consider dependence with memory
- **($11, i1, i5)**
- **($24, i2, i3)**
- **($24, i3, i5)**

3.2(5 pts) If there is no forwarding nor reordering, we need to insert nops /pipleline stalls to ensure correct execution. Suppose we report the number of stalls inserted between consecutive instructions in the format of `(number_of_nops,`

`predecessor_instruction, successor_instruction)`. For example: `(3, I10, I11)` specifies that `3` nops need to be inserted between instruction `I10` and `I11`. Fill in the blanks below to give the correct number of nops / stalls for the given sequence of instructions.

`(0, I1, I2)`

`(2, I2, I3)`

`(0, I3, I4)`

`(1, I4, I5)`

> **Note:** if no stall needed, answer with 0

3.3 (11pts) If there is full forwarding support into the `EX` stage, **draw** multiple-cycled pipeline diagram to show the execution of the given 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.

- Textbook example of multiple-cycled pipeline diagram: **Figure 4.43 / Figure 4.44**
- Textbook example of pipeline diagram with forwarding: **Figure 4.53 / Figure 4.59**
- **i1**
- **i2**
- **nop**
- **i3**
- **i4**
- **i5**

4. (10 pts) Branch predictors. Suppose we define the accuracy of a predictor to be (the number of correct predictions / the number of total predictions). Given the branch sequence as **NT, T, T, T, NT, T, T, T.**

4.1. (8 pts) Fill in the table below to show the status transition/prediction of a two-bit predictor for the given branch sequence. Assume that the predictor starts off in the top right state from **Figure 4.63** of textbook ((weak) predict taken).

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

4.2. (2 pts) What is the accuracy of this two-bit predictor based on your table above? What is its accuracy if the same branch sequence repeats forever? Note: include a brief explanation for each number.

- Error rate = no. of errors / no. of predictions
- no. of errors = 2
- no. of predictions = 8
- Error rate = 2/8 = ¼
- Accuracy = 1-Error rate = 0.75 or 75%

Part 2. MIPS Programming (50%)

MIPS Scheduler. 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 true data dependences as well as stalls triggered by data hazards for a 5-stage pipeline processor with ***no forwarding***.

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)
- Download and use the provided **cs465_hw3_template.asm** as your starting point. It already includes MIPS code that accepts/verifies `N` and a sequence of `N` instructions from the standard input.

Decode each instruction 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 `I1`, the second instruction gets label `I2`, etc.
- You only need to support a subset of MIPS instructions: `sub, addi, slt, lw, sw, bne`. (We drop `j` and `jal` from the set of instructions of HW2 for this assignment.)
- You can assume all instructions from the user are valid instructions within the subset of MIPS ISA that we support.
- The provided **cs465_hw3_template.asm** includes MIPS code that call `atoi()` to convert the input string into a number and save the value in an array pre-allocated with 10 entries. You will need to provide a working `atoi()` implementation, which has the same definition as HW2. Feel free to reuse your code from the previous assignment.

Identify and **report** the instruction(s) that each instruction is data-dependent on.

- If a register read by an instruction (source register) is last updated by an earlier instruction in the given sequence, we need to list this register as well as the producer instruction. See sample runs below for format details.
- If there are more than one true dependence for an instruction, report the dependence involving **rs first** followed by the dependence involving **rt**. However, if an instruction uses the same register as both rs and rt, report the dependence only once.
- Only report true data-dependence – no report of anti-dependence nor output dependence. You will need to report all flow dependence include those that will not cause any data hazards / delay of pipelined execution.
- We ignore the control flow caused by branches, which means that you only need to analyze the given instructions in sequential order.
- Your implementation of `get_src_regs()` from HW2 could be helpful. Feel free to define additional helper functions.
- The provided **cs465_hw3_template.asm** includes a helper function `print_dependence()` that you can use for output purpose.

Schedule the given sequence of instructions and **report** at which cycle we can start the execution of each instruction.

- We assume a 5-stage pipeline processor **with no forwarding**.
- The issue cycle starts with Cycle 1.
- If there are no data hazards, we always start one instruction per cycle.

- If any true dependence causes data hazards, we will need to insert appropriate number of stall cycles (i.e. pipeline bubbles) to ensure correct execution.
  - **Note**: not all data-dependences will cause data hazards. It is possible for an instruction to be data-dependent on prior instructions but no stall is needed.
- Again, no need to consider control flows or control hazards.
  - Process the given instructions in sequential order only.
  - Ignore control hazards; you only need to consider stalls caused by data hazards.
- The provided **cs465_hw3_template.asm** includes a helper function `print_cycle()` that you can use for output purpose. Check the Appendix for examples.

Coding Requirements and Suggestions:

You must use the provided template to accept input from the user.

You code must be very well commented. A description of the algorithm you use must be included in your comments.

Feel free to define additional helper functions and/or macros.

You might find the provided or your own MIPS code from HW1 and/or HW2 helpful. Feel free to use them for this assignment.

System calls: http://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html

Macros: https://courses.missouristate.edu/KenVollmar/mars/Help/MacrosHelp.html

Grading rubric:

Code submission format and documentation	5/50
Code assembled with no error	5/50
Dependence report	25/50
Schedule start cycles	15/50

We will be comparing the generated output file and the expected output file for grading.

No late work allowed after 24 hours.

- Late submission automatically uses one of your tokens. If you run out of tokens, late penalty will be applied.

Extra credit for early submissions:

- 1% extra credit rewarded for every 24 hours your submission made before the due time.
- Up to 3% extra credit will be rewarded.
- Your latest submission will be used for grading and extra credit checking. You CANNOT choose which one counts.

You will need to make two separate submissions for Part1 and Part2 respectively. The later one of those two will be used to determine whether your HW3 is early / normal/ late.

Appendix

MIPS instructions that you must support:

You are required to support only a subset of MIPS instructions. Use the table below as your reference. ~~For all other inputs, your program should return/print "invalid".~~ For HW3, you can assume only valid instruction from the table below will be used in input sequences.

MIPS	insn_code	opcode	funct	MIPS	insn_code	opcode	funct
sub	0x0	0x00	0x22	sw	0x4	0x2b	-
addi	0x1	0x08	-	bne	0x5	0x05	-
slt	0x2	0x00	0x2a	j (not for HW3	~~0x6~~	~~0x02~~	~~-~~
lw	0x3	0x23	-	~~jal~~ (n)ot for HW3	~~0x7~~	~~0x03~~	~~-~~

Arrays in MIPS:

You may want to define arrays in your program. Here are some MIPS basics regarding arrays.

- Array declarations. Arrays can be allocated using labels in a data segment.

```
.data
   myArray: .space 12   # myArray is an array of 12 bytes, which can be used
                        # as an array of 3 words (each 4 bytes)
                        # int myArray[3];

   myInts: .word 100:5 # myInts is an array of 5 words/integers,
                        # each is initialized as 100
                        # int myInts[5] = {100, 100, 100, 100, 100};

   .align 2                    # make sure align with 2^2 (4 bytes)
   nums: .word 5, 14, 15       # nums is an array of 3 words/intgers,
                               # they are initialized to be 5, 14, and 15
                               # int nums[3] = {5, 14, 15};
```

- Array accesses. We need to specify the byte address to access an array element. There are multiple ways to do that. See examples below.

```
.text
   la $t0, nums          # load starting address of nums in $t0
   sw $t1, 4($t0)        # set nums[1] = $t1

.text
   li $t2, 8             # $t2 = 8
   lw $t2, myInts($t2) # read memory $t2 = myInts[2]
                         # note label myInts can be used as a constant
```

Check the provided template file to see more examples of array definitions and accesses.

Data dependence and delay examples:

You need to identify and report true data dependences only.

- **Example 1:**

	Instruction	Source Register(s)	Dependence (Producer Instruction)	Start Cycle	Notes
I1	`sub $t8,$t9,$t2`	$t9	none	1	1st instruction starts at cycle 1
		$t2	none		
I2	`sub $t9,$t8,$t2`	$t8	I1	4	2-cycle stall (cycle 2/3) for I1 to pass $t8 to I2
		$t2	none		
I3	`sub $t3,$t8,$t9`	$t8	I1	7	2-cycle stall (cycle 5/6) for I2 to pass $t9 to I3
		$t9	I2		
I4	`sub $t8,$s7,$t8`	$s7	none	8	No data hazards although there is a dependence: no stall
		$t8	I1		

- I1 and I4 have an output dependence with $t8: no report
- I1 and I2 have an anti-dependence with $t9: no report

- **Example 2:**

	Instruction	Source Register(s)	Dependence (Producer Instruction)	Start Cycle	Notes
I1	`lw $t8,0($t9)`	$t9	none	1	1st instruction starts at cycle 1
I2	`sub $t7,$t8,$t2`	$t8	I1	4	2-cycle stall (cycle 2/3) for I1 to pass $t8 to I2
		$t2	none		
I3	`sub $s7,$s7,$t2`	$s7	none	5	No flow dependence
		$t2	none		
I4	`sub $t3,$t8,$t7`	$t8	I1	7	Need a 2-cycle gap (cycle 5/6) from I2
		$t7	I2		

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

- **Example 3:**

	Instruction	Source Register(s)	Dependence (Producer Instruction)	Start Cycle	Notes
I1	`sub $t0,$t1,$t2`	$t1	none	1	1st instruction starts at cycle 1
		$t2	none		
I2	`sub $t0,$t3,$t0`	$t3	none	4	2-cycle stall (cycle 2/3) for I1 to pass $t0 to I2
		$t0	I1		
I3	`bne $t0,$zero,I1`	$t0	I2	7	I3 depends on I2 not I1 since I2 occurs later in time; 2-cycle stall (cycle 5/6) for I2 to pass $t0 to I3
		$0	none		

  - I1: no report of $t0 flows from I2 back to I1 since `I1->I2->I3->I1` only happens if branch of I3 is taken, which we ignore for this assignment.

- **Example 4:**

	Instruction	Source Register(s)	Dependence (Producer Instruction)	Start Cycle	Notes
I1	`sub $t0,$t1,$t2`	$t1	none	1	1st instruction starts at cycle 1
		$t2	none		
I2	`bne $t3,$zero,I4`	$t3	none	2	No dependence
		$0	none		
I3	`sub $t2,$t4,$t0`	$t4	none	4	Need a 2-cycle gap (cycle 2/3) from I1
		$t0	I1		

  - I3: need to report $t0 flows from I1 since we assume sequential execution (i.e. the branch at I2 is not taken).

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

Note: Below are multiple sample runs. Newline and user input display might be different depends on whether you run directly in a prompt/terminal or with MARS GUI (and your setting with MARS) but all numbers/strings should match.

```
How many instructions to process (valid range: [1,10])? 4↵
Please input instruction sequence (one per line):
Next instruction: 0x032AC022↵
Next instruction: 0x030AC822↵
Next instruction: 0x03195822↵
Next instruction: 0x02F8C022↵
I1
Dependence Info: None
Start Cycle: 1
 ----------------------------------------
I2
Dependence Info:
      Source Register: 24, Producer Instruction: I1
Start Cycle: 4
 ----------------------------------------
I3
Dependence Info:
      Source Register: 24, Producer Instruction: I1
      Source Register: 25, Producer Instruction: I2
Start Cycle: 7
 ----------------------------------------
I4
Dependence Info:
      Source Register: 24, Producer Instruction: I1
Start Cycle: 8
 ----------------------------------------
```

```
#Example 1 from Appendix #I1:sub $t8, $t9, $t2 #I2:sub $t9, $t8, $t2 #I3:sub $t3, $t8, $t9 #I4:sub $t8, $s7, $t8
```

```
# report dependence with # $rs, followed by
# dependence with $rt
```

```
How many instructions to process (valid range: [1,10])? 4↵
Please input instruction sequence (one per line):
Next instruction: 0x8F380000↵
Next instruction: 0x030A7822↵
Next instruction: 0x02EAB822↵
Next instruction: 0x030F5822↵
I1
Dependence Info: None
Start Cycle: 1
 ----------------------------------------
I2
Dependence Info:
      Source Register: 24, Producer Instruction: I1
Start Cycle: 4
 ----------------------------------------
I3
Dependence Info: None
Start Cycle: 5
 ----------------------------------------
I4
Dependence Info:
```

```
#Example 2 from Appendix #I1: lw $t8, 0x0($t9) #I2: sub $t7, $t8, $t2 #I3: sub $s7, $s7, $t2 #I4: sub $t3, $t8, $t7
```

```
      Source Register: 24, Producer Instruction: I1
      Source Register: 15, Producer Instruction: I2
Start Cycle: 7
 ----------------------------------------
```

```
How many instructions to process (valid range: [1,10])? 3↵
Please input instruction sequence (one per line):
Next instruction: 0x012A4022↵
Next instruction: 0x01684022↵
Next instruction: 0x1500FFFD↵
I1
Dependence Info: None
Start Cycle: 1
 ----------------------------------------
I2
Dependence Info:
      Source Register: 8, Producer Instruction: I1
Start Cycle: 4
 ----------------------------------------
I3
Dependence Info:
      Source Register: 8, Producer Instruction: I2
Start Cycle: 7
 ----------------------------------------
```

```
#Example 3 from Appendix # I1: sub $t0, $t1, $t2 # I2: sub $t0, $t3, $t0 # I3: bne $t0, $zero, I1
```

```
How many instructions to process (valid range: [1,10])? 3↵
Please input instruction sequence (one per line):
Next instruction: 0x012A4022↵
Next instruction: 0x15600001↵
Next instruction: 0x01885022↵
I1
Dependence Info: None
Start Cycle: 1
 ----------------------------------------
I2
Dependence Info: None
Start Cycle: 2
 ----------------------------------------
I3
Dependence Info:
      Source Register: 8, Producer Instruction: I1
Start Cycle: 4
 ----------------------------------------
```

```
#Example 4 from Appendix # I1: sub $t0, $t1, $t2 # I2: bne $t3, $zero, I4 # I3: sub $t2, $t4, $t0 # I4: ...
```