

Computer Engineering 431, Spring 2018 Exam 2, Thursday, March 1st

Exam time: 5 pre + 70 minutes exam

Test Value: 32 pts. Total possible points: 40 (max score = 125%)

Total =	/32	P1:	/8	P2:	/10	P3:	/10	P4:	/8	P5:	/4
---------	-----	-----	----	-----	-----	-----	-----	-----	----	-----	----

Front Left Neighbor # _____	Front Neighbor # _____	Front Right Neighbor # _____
Left Neighbor # _____ _____	Your Name (print clearly): _____ _____	Right Neighbor # _____ _____
Rear Left Neighbor # _____	Rear Neighbor # _____	Rear Right Neighbor # _____

1. Dependencies (8 pts)

- a) (4) The code segment below, `Aardvark(TreePtr, SearchVal)`, returns the number of elements in a binary tree that match `SearchVal`.

```

1| Aardvark:      add $v0, $0, $0; set return value
2|   bne $a0, $0, Badger      ; check for null ptr
3|   jr $ra                      ; return $v0
4| Badger:  lw $t0, 0($a0)      ; load current value
5|   bne $t0, $a1 Capybara; do not count non-matches
6|   addi $v0, $v0, 1 ; Match, increment return value
7| Capybara:      addi $sp, $sp, -16 ; reserve stack space
8|   sw $a0, 12($sp) ; store argument 0
9|   sw $a1, 8($sp) ; store argument 1
10|  sw $ra, 4($sp) ; store return address
11|  sw $v0, 0($sp) ; store current total
12|  lw $a0, 4($a0) ; load left branch ptr into arg 0
13|  jal Aardvark ; recurse left
14|  lw $t0, 0($sp) ; restore previous total
15|  add $v0, $v0, $t0 ; sum with left branch
16|  sw $v0, 0($sp) ; store current total
17|  lw $a0, 12($sp) ; restore arg 0
18|  lw $a1, 8($sp) ; restore arg 1
19|  lw $a0, 8($a0) ; load right branch ptr into arg 0
20|  jal Aardvark ; recurse right
21|  lw $ra, 4($sp) ; restore return address
22|  lw $t0, 0($sp) ; retrieve previous total
23|  add $v0, $v0, $t0 ; sum right total with previous
24|  addi $sp, $sp, 16 ; release stack space
25|  jr $ra ; return $v0

```

Assuming that branch delay slots are not present, how many basic blocks (single-entry, single-exit regions) are there in the control flow graph for the above function, and which instruction ranges belong to which block? (**note: there will be LESS THAN 15 blocks**)

B1: to	B2: to	B3: to	B4: to	B5: to
B6: to	B7: to	B8: to	B9: to	B10: to
B11: to	B12: to	B13: to	B14: to	B15: to

- b) (4) Label all data dependencies and (false/anti)-dependencies in the below code fragment with an arrow and their data dependency type.

```

1| FOO: lw      $2, 0($4)
2| lw          $3, 0($5)
3| addu        $2, $2, $3
4| sw          $2, 0($4)
5| lw          $4, 4($4)
6| addi        $5, $5, 4
7| bne         $4, $0, FOO

```

2. Pipelining with data hazards (10 pts)

- a) (7) Consider the instructions from 1(b) scheduled on a five stage, scalar MIPS pipeline (FDEMW) **with full forwarding and bypass networks and hazard detection and branch resolution in D**. In the table below, for each cycle, indicate the cycle an instruction completes a stage with a capital letter (FDEMW) and indicate stalls with a lowercase letter (fdemw). **Indicate by drawing a vertical arrow when a value is bypassed from one instruction to another in the cycle that the forwarding occurs.** For simplicity, omit W→D bypassing arrows. Assume all loads/stores are 1-cycle hits, that there are no exceptions, and that all branches are perfectly predicted.

CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
FOO: lw \$2, 0(\$4)																			
lw \$3, 0(\$5)																			
addu \$2, \$2, \$3																			
sw \$2, 0(\$4)																			
lw \$4, 4(\$4)																			
addi \$5, \$5, 4																			
bne \$4, \$0, FOO																			

- b) (3) Given the schedule below, list what forwarding is demonstrated as being supported (and list the pair of instructions that demonstrate this, for each pipeline stage pair) and where hazard detection and branch resolution occur.

CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
FOO: lw \$2, 0(\$4)	F	D	E	M	W														
lw \$3, 0(\$5)		F	D	E	M	W													
addu \$2, \$2, \$3			F	D	e	E	M	W											
sw \$2, 0(\$4)				F	d	D	e	E	M	W									
lw \$4, 4(\$4)					f	F	d	D	E	M	W								
addi \$5, \$5, 4							f	F	D	E	M	W							
bne \$4, \$0, FOO									F	D	E	M	W						

3. Superscalar In-order Processors (10 pts)

- a) (4) Assume that the pipeline in 2(a) is extended to be 2-wide and all functional units are replicated 2x, including memory ports. Schedule the below instructions for this pipeline, assuming that the branch to FOO is both taken and correctly predicted and that no other instruction can advance into a pipeline stage where any instruction is currently stalled.

CYCLE → Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
FOO: lw \$2, 0(\$4)	F																		
lw \$3, 0(\$5)	F																		
addu \$2, \$2, \$3																			
sw \$2, 0(\$4)																			
lw \$4, 4(\$4)																			
addi \$5, \$5, 4																			
bne \$4, \$0, FOO																			
FOO: lw \$2, 0(\$4)																			









- b) (2) Assume that multiple copies of the same code were running on an FGMT (Fine-Grained-Multi-Threaded) processor with simple round-robin scheduling. What is the minimum number of contexts that the FGMT processor would need to support in order to remove all data dependency stalls for the above code fragment?

- c) (4) Consider a 2-wide VLIW system where the only slot restriction is that at most 1 control operation can be present in a bundle, but the pipeline does not support any forwarding other than $W \rightarrow D$ and has no hazard detection for data dependencies. **While preserving identical functionality**, rewrite the code from 2(a) as VLIW bundles and then schedule them, including any NOPs and entirely NOP bundles. Assume that registers \$2 to \$30 are freely available for use in your new instructions, but that live-in register names are the same as in the original code.

CYCLE → Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	F	D	E	M	W														
	F	D	E	M	W														

4. Branch prediction (8 pts)

- a) (1) Why is stalling, used for data dependencies, insufficient to also solve control dependencies?
- b) (1) In class, we discussed a 2-wide ARM processor design where branch resolution is very late in a deep pipeline. If 25% of your instructions are branches, branch resolution is in the 4th pipeline stage, and your prediction is 75% accurate, what is the expected (additive) increase in CPI due to branch mispredictions (over perfect prediction)?
- c) (2) Assume that a given static branch has a repeating pattern of TTTN. Going from a 1-bit bimodal predictor to a 2-bit bimodal predictor increases prediction accuracy by 25% (steady state). What are the gains for this pattern from using 3-bit bimodal predictor? What common feature of all three prediction schemes described by Yeh & Patt is fundamental in achieving higher accuracy for the above scenario with their predictors than with a bimodal predictor?
- d) (4) 2-bit bimodal predictors are generally better than 1-bit bimodal predictors, but both can be subject to pathological behaviors. Given a 1-bit and a 2-bit bimodal predictor, both starting in their weakest not-taken state, fill in all (22) empty boxes to show a length 8 sequence of branch outcomes, and the associated predictions, wherein each predictor mispredicts only the circled entries, resulting in the 1-bit predictor having a higher accuracy than the 2-bit predictor.

Actual Branch Outcomes (N,T) →								
1-bit prediction state (N,T)	N							
2-bit prediction state (N,n,t,T)	n							

5. Caching in Virtual Memory Systems [revisited] (4 pts)

a) (1) The register transfer logic for “ADDU \$rd, \$rs, \$rt” is written $REG[\$rd] = REG[\$rs] + REG[\$rt]$ considering the register file as the array REG. If we additionally consider memory as the byte-array MEM, what is the register transfer logic for “LW \$rt, imm (\$rs)”?

b) (3) Assume that you have a system with the following properties and configuration

(Same values as exam 1):

- The system is byte-addressable with 16-bit words
 - Physical address space: 10 bits; Virtual address space: 16 bits; Page size: 2^4 bytes
 - VIPT L1 D-cache = 48 bytes, 3-way associative, with 8-byte blocks; write-allocate/write-back
 - Fully associative 3 entry DTLB; both DTLB and L1 D-Cache use LRU policy.
 - Entry for each TLB or Cache entry consists of {valid, dirty, LRU-rank (00=most recent), tag, data}
- All metadata is given in binary. TLB data is in binary and Cache data is in hexadecimal.
- Endianness: If the data block containing address 0x0006 was 0x0123456789ABCDEF, the word loaded from 0x0006 would have integer value = 0xCDEF.

DTLB:

1,0,00, 0000 1000 0000, 00 1100	1,0,10, 1101 0000 1101, 00 0001	1,0,01, 0001 1101 0000, 01 1111
------------------------------------	------------------------------------	------------------------------------

L1 D-Cache:

SET 0	1, 0, 01, 00 1100, 0x1234567887654321	1, 0, 00, 00 0001, 0xCD9CEF0990FEDCBA	0, 0, 10, 01 1001, 0xBAD1BAD2BAD3BAD4
SET 1	1, 0, 10, 01 1111, 0xFEEDD0D0EEC5F00D	1, 0, 00, 00 1100, 0x1FEEDEE15DEADC0D	1, 0, 01, 00 0001, 0x0102030405060708

Given the initial contents of the DTLB and L1 D\$ as shown, fill in the register value after the instruction executes.

LW \$1, 0x806(\$0); REGISTER_FILE[\$1]=0x_____