EAST WEST UNIVERSITY

Assignment -2

Course Info:

Course Name: Computer Architecture

Course Code: CSE360

Section: 01

Semester: Spring 2023

Course Instructor:

Md. Ezharul Islam

PhD Professor

Department of Computer Science and Engineering

Students Info:

Name: Md. Sabbir Hossain

ID: 2020-2-60-107

4.8.2

Pipelined	Single-cycle
350 ps	1250 ps

4.8.2

Pipelined	single-cycle
1750 ps	1250 ps

4.8.3

Stage to split	New Clock Cycle Time
ID	300 ps

4.8.4

a.	35%

4.8.5

a.	65%

4.8.6

We already computed clock cycle times for pipelined and single cycle organizations, and the multi-cycle organization has the same clock cycle time as pipelined organization. We'll compute execution times relative to the pipelined organization. In single-cycle, every instruction takes one (long) clock cycle. In pipelined, a long-running program with no pipeline stalls completes one instruction in every cycle. Finally, a multicycle organization completes a LW in 5 cycles, a SW in 4 cycles (no WB), an ALU instruction in 4 cycles (no MEM), and a BEQ in 4 cycles (no WB). So we've the speedup of pipeline.

Multi cycle: execution time is X times pipelined execution time, where X is:	Single-cycle execution time is X times pipelined execution time, where X is:
0.20*5+0.80*4 = 4.20	1250ps / 350ps = 3.57

4.10

4.10.1 In the pipelined executing shown below, *** represents a stall When an instruction can't be fetched because a load or store instruction is using the memory in that cycle. cycles are represented from left to right, and for each instruction we show the pipeline stage it is in during that cycle:

Instruction	Pipeline stage	cycle
SW R16, 12 (R6)	IF ID EX MEM WB	
LW R16, 8 (R6)	IF ED EX MEM WB	
BEQ R3, R4, Lb]	IF ID EX MEM WB	11
ADD R3, R1, R4	*** *** IF ID EX MEM WB	
SLT R3, R13, R4	IF ID EX MEM WB	

4.10.3 Stall-on-branch delays the fetch of the next instruction until the branch is executed. When branches execute in the EXE stage, each branch causes two stall cycles. When branches execute in the ID stage, each branch only causes one stall cycle. Without branch stalls (e.g., with perfect branch prediction) there're no stalls, and the execution time is 4 plus the number of executed instructions. We've:

Instructions Executed	Branches Executed	cycles with branch in EXE	cycles with branch in ID	Speedup
5	1	4+5+1*2 =11	4+5+1*1 =10	11/10 =1.10

4.10.5

New ID latency	New EX latency	New cycle time	old cycle time	speed-up
180ps	140 ps	200ps (IF)	200ps (IF)	(11*200) /(10*200) =1.10

4.10.6

The cycle time remain uncharged: a 20 ps reduction is Ex latency has no effect on clock cycle time because Ex is not the longest-latency stage. The change does affect execution time because it adds one additional stall cycle to each branch. Because the clock cycle time doesn't improve but the number of cycle increases, the speedup from this change will be below 1 (a slowdown). In 4.10.3 we already computed the number of cycles when branch is in EX stage

4.10.4

The number of cycles for the (normal) 5 stage and the (combined EX/MEM) 4 stage pipeline is already computed in 4.10.2. The clock cycle time is equal to the latency of the longest-latency stage. Combining EX and MEM stages affects clock time only if the combined EX/MEM stage becomes the longest-latency stage:

cycle time with 5 stages	cycle time with 4 stages	speedup
200 ps (IF)	210ps (MEM + 20 ps)	(9* 200)/(10*200 = ~~1.10~~ 1.07

We've:

	cycles with branches in EX	Execution time (branch in EX)	cycles with branch in MEM	Execution time (branch in MEM)	Speedup
a.	4+5+ 1*2=11	11* 200ps = 2200ps	4+5+ 1*3 =12	12*200ps = 2400ps	0.92

4.11

4.11.1

```
LW R1,0(R1)
LW R1,0 (R1)
BEQ R1,R0, Loop
LW R1,0(R1)
AND R1,R1,R2
LW R1,0(R1)
LW R1,0(R1)
BEQ R1,R0,Loop
```

```
WB
EX  MEM WB
ID  ***  EX  MEM WB
IF  ***  ID  EX  MEM WB
         IF  ID  ***  EX MEM WB
             IF  ***  ID EX MEM
                  IF  ID  ***
                      IF  ***
```

4.11.2 In a particular clock cycle, a pipeline stage is not doing useful work if it's ~~co~~ stalled or if the instruction going through that stage is not doing any useful work there. In the pipeline execution diagram from 4.11.1, a stage is stalled if it's name is not ~~foun~~ shown for a particular cycles, and stages in which ~~the~~ particular instruction is not doing useful work are marked in blue. Note that a BEQ instruction is doing useful work in the MEM stage, because it's determining the correct value of the next instruction's PC in that stage.

We've:

cycle per loop iteration	cycles in which all stages do useful work	% of cycles in which all stages do useful work
8	0	0%

4.13.3

With forwarding, the hazard detection unit is still needed because it must insert a one-cycle stall whenever the load supplies a value to the instruction that immediately follows the load. Without the hazard detection unit, the instruction that depends on the immediately preceding load gets the stale value the register had before the load instruction.

---

code executes correctly (for both loads, there is no RAW dependence between the load and the next instruction).

4.13.4 The outputs of the hazard detection unit are PCWrite, IF/ID Write, and ID/EXzero (which controls the Mux after the output of the control unit). Note that IF/ID Write is always equal to PC Write, and ED/EX zero is always the opposite of PCWrite. As a result, we'll only show the value of PCWrite for each cycle. The outputs of the forwarding unit is ALUin1 and ALUin2, which control Muxes that select the first and second input of the ALU. The three possible value of for ALUin1 or ALUin2 are 0 (no forwarding), 1 (forward ALU output from previous instruction), or 2 (forward data value for second - previous instruction)

We have:

4.13.5 The instruction that's currently in the ID stage needs to be stalled if it depends on a value produced by the instruction in the EX or the instruction in the MEM stage. SO we need to check the destination registers of these two instructions. For the instructions in the EX stage, we need to check Rd for R-type instructions and Rd for loads. For the instruction in the MEM stage, the destination register is already selected (by the Mux in the EX stage) so we need to check that register number (this is the bottomost output of the EX/MEM Pipeline registers). The additional inputs to the hazard detection unit are register and output number of the output register from the EX/MEM pipeline register. The Rt field from the ID/EX register is already an input of the hazard detection unit in Figure 4.60.
No additional outputs are needed. We can sta

between the value read from Registers, the ALU output from the EX/MEM pipeline register, and the data value from the MEM/WB pipeline register. The complexity of the new forwarding unit is the same as the complexity of the existing one.

4.15

4.15.1 Each branch that's not correctly predicted by the always-taken predictor will cause 3 stall cycles, so we've:

Extra CPI
$3*(1-0.45)*0.25 = 0.41$

4.15.2 Each branch that not correctly predicted by the always-not-taken predictor will cause 3 stall cycles, so we've:

Extra CPI
$3*(1-0.55)*0.25 = 0.34$

4.15.3 Each branch that not correctly predicted by the 2-bit predictor will cause 3 stall cycles, so we've:

Extra CPI
$3*(1-0.85)*0.25 = 0.113$

4.15.4 Correctly predicted branches had CPI of 1 and now they become ALU instructions whose CPI is also 1. Incorrectly predicted instructions that are converted also become ALU instructions with a CPI of 1, so we've:

CPI without conversation	CPI with conversation	Speedup from conversation
$1+3*(1-0.85)*0.25 = 1.113$	$1+3*(1-0.85)*0.25*0.5 = 1.056$	$1.113/1.056 = 1.054$

4.15.5 Every converted branch instruc-tion now takes an extra cycle to exe-cute, so we've:

CPI without conversation	CPI with conversation	speedup from conversation
1.113	1+(1+3*(1-0.85)* 0.25*0.5 = 1.181	1.113/1.181 = 0.94

4.15.6
Let the total number of branch instructio-n executed in the program be B. We have:

correctly Predicted	correctly predic-ted non-loop-back	Accuracy on on-loop-back branches
B*0.85	B*0.05	(B*0.05/(B*0.20) = 0.25 (25%)

4.16
4.16.1

Always Taken	Always Non-taken
3/5 = 60%	2/5 = 40%

there is no warmup period for the given pattern, the warm-up period for the opposite pattern is only one branch.

4.17

4.17.1

Instruction 1	Instruction 2
Invalid target address (EX)	Invalid data address (MEM

4.17.2

The Mux that selects the next PC must have inputs added to it. Each inputs is a constant address of an exception handlers. The exception detectors must be added to the appropriate pipeline stage and the outputs of these detectors must be used to control the pre-PC Mux, also to convert to NOPs instruc-tions that are already in the pipeline behind the exception triggering instruction.

Date :

4.19.4

Before the change, the control unit decodes the instruction while register reads are happening. After the change, the latencies of control and Register Read can't be overlapped. This increases the latency of the ID stage and could affect the processor's clock cycle time if the ID stage becomes the longest-latency stage.
We have:

Clock cycle time before change	Clock cycle time after change
250ps (D-Mem in MEM stage)	No change (150ps + 90ps < 250ps)

Date :

3.3.1

Assuming the address given as byte addresses, each group of 16 accesses will map to the same 32-byte block so the cache will have a miss rate of 1/16. All misses are compulsory misses. The miss rate isn't sensetive to the size of the cache or the size of the working set. It's, however, sensitive to the access pattern and block size.

3.3.2

The miss rates are 1/8, 1/32, and 1/64, respectively. The workload is exploting temporal locacity.

3.3.4

AMAT for B = 8: 0.040 x (20x8) = 6.40
AMAT for B = 16: 0.030 x (20x16) = 9.60
AMAT for B = 32: 0.020 x (20x32) = 12.80
AMAT for B = 64: 0.015 x (20x64) = 19.20
AMAT for B = 128: 0.010 x (20x128) = 25.60

B = 8 is optimal

Date :.....

5.5.5

AMAT for B = 8: 0.040 × (24 + 8) = 1.28

AMAT for B = 16: 0.030 × (24 + 16) = 1.20

AMAT for B = 32: 0.020 × (24 + 32) = 1.12

AMAT for B = 64: 0.015 × (24 + 64) = 1.32

AMAT for B = 128: 0.010 × (24 + 128) = 1.52

B = 32 is optimal.

5.6.6

P1 AMAT = 0.66 ns + 0.08 × 70 ns = 6.26 ns

P2 AMAT = 0.90 ns + 0.06 × (5.62 ns + 0.95 × 70 ns) = 5.23 ns

For P1 to match P2's performance:

5.23 = 0.66 ns + MR × 70 ns

MR = 6.5%

Date :

5.7.2

since this cache is fully associative and has ~~one~~ one word blocks, the word address is equivalent to the tag. The only possible way for there to be a hit is a repeated reference to the same word, which doesn't occur for this sequence.

Tag	Hit / Miss	Contents
3	M	3
180	M	3, 180
43	M	3, 180, 43
2	M	3, 180, 43, 2
191	M	3, 180, 43, 2, 191
88	M	3, 180, 43, 2, 191, 88
190	M	3, 180, 43, 2, 191, 88, 190
14	M	3, 180, 43, 2, 191, 88, 190, 14
181	M	181, 44, 43, 2, 191, 88, 190, 14
44	M	181, 44, 43, 2, 191, 88, 190, 14
186	M	181, 44, 186, 2, 191, 88, 190, 14
253	M	181, 44, 186, 253, 191, 88, 190, 14

5.7.3

Address	Tag	Hit/Miss	contents
3	1	M	1
180	90	M	1, 90
43	21	M	1, 90, 21
2	1	H	1, 90, 21
191	95	M	1, 90, 21, 95
88	44	M	1, 90, 21, 95, 44
190	95	H	1, 90, 21, 95, 44
14	7	M	1, 90, 21, 95, 44, 7
181	90	H	1, 90, 21, 95, 44, 7.
44	22	M	1, 90, 21, 95, 44, 7, 22
186	143	M	1, 90, 21, 95, 44, 7, 22, 143
253	126	M	1, 90, 126, 95, 44, 7, 22, 143

Date :.....

The final reference replaces tag 21 in the cache, since tags 1 and 90 had been reused at time = 3 and time = 8 while 21 hadn't been used since time = 2.
Miss rate = 9/12 = 75%

This is the best possible miss rate, since there were no misses on any block that had been previously evicted from the cache. Infact, the only eviction was for tag 21, which is only referenced once.

5.7.4

L1 only

$.07 \times 100 = 7m$

CPI = 7m/.5m = 14

Direct mapped L2:

$.07 \times (12 + 0.035 \times 100) = 1.1m$

CPI = ceiling (1.1m/.5m) = 3

8-way set associated L2:

$.07 \times (28 + 0.015 \times 100) = 2.1m$

CPI = ceiling (2.1m/.5m) = 5

Double memory access time, L1 only:

$.07 \times 200 = 14m$

CPI = 14m/.5m = 28

Double memory access time, direct mapped L2:

$.07 \times (12 + 0.035 \times 200) = 1.3m$

CPI = ceiling (1.3m/.5ns) = 3

Double memory access time, 8-way set associated L2:

$.07 \times (28 + 0.015 \times 200) = 2.2m$

CPI = ceiling (2.2m/.5m) = 5

Halved memory access time, L1 only:

$.07 \times 50 = 3.5m$

CPI = 3.5m /. 5m = 7

Halved memory access time, direct mapped L2:

$0.7 \times (12 + 0.035 \times 50) = 1.0m$

CPI = ceiling (1.1m /. 5m) = 2

Halved memory access time, 8 way set associated L2:

Date :

~~0.7~~ .07 x (28 + 0.015 x 50) = 2.1 ns

CPI = ceiling (2.1ns / .5ns) = 5

5.7.5

.07 x (12 + 0.035 x (50 + 0.013 x 100)) = 1.0 ns

Adding the L3 cache does reduce the overall memory access time, which is the main advantage of having a L3 cache. The disadvantage is that the L3 cache takes real estate away from having other types of resources, such as functional units.

5.7.6

Even if the miss rate of the L2 cache was 0, a 50 ns access time gives AMAT = .07 x 50 = 3.5 ns, which is greater than the 1.1 ns and 2.1 ns given by the on-chip L2 caches. As such, no size will acheive the performance goal.