



The University of Michigan - Department of EECS

EECS370 – Introduction to Computer Organization

Midterm Exam 2
April 3, 2007

Name: _____ **KEY (answers are in red)** _____

University of Michigan unickname: _____
(NOT your student ID number!)

Open book, open notes. No laptops, PDAs, cell phones, etc. (calculators are ok). This exam has 9 questions, 12 pages, and 100 points. Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question. For questions where a box is provided, please put your final answer in the box.

The rules of the Honor Code of the University of Michigan - College of Engineering apply for this exam.

Honor code pledge: I have neither given nor received aid on this examination, nor have I concealed any violations of the Honor Code.

Signature: _____
(Exams without a signed pledge will not be graded)

1. Short questions [13 points]

(a) [1 pt] **TRUE** FALSE A predict-taken branch predictor is more difficult/expensive to build than a predict-not taken predictor. Circle TRUE or FALSE.

(b) [1 pt] **TRUE** FALSE The number of stalls caused by load-use data hazards can be reduced by intelligently reordering instructions. Circle TRUE or FALSE.

(c) [1 pt] TRUE **FALSE** Program execution time on the LC2k multi-cycle datapath can be reduced by using a branch predictor. Circle TRUE or FALSE.

(d) [1 pt] TRUE **FALSE** Changing a direct-mapped cache to a 2-way associative cache is likely to reduce the number of capacity misses. Circle TRUE or FALSE.

(e) [3 pts] Given the following: 16-bit address, byte addressable, cache size is 64 bytes, block size is 4 bytes, tag size is 12 bits, what is the associativity of the cache? Write your answer in the box.

4-way

(f) [3 pts] We want to extend *Project 3* with an additional pipeline to make it a 2-way super-scalar. In the project, there were 6 data forwarding paths (EX/MEM to ALUin1, EX/MEM to ALUin2, MEM/WB to ALUin1, MEM/WB to ALUin2, WB/END to ALUin1, WB/END to ALUin2). How many forwarding paths will there be in the 2 pipeline case? Justify your answer with one sentence.

24

(g) [3 pts] Consider the single cycle (SC), multi-cycle (MC), and pipelined (P) LC2k datapaths that were discussed in class. Using the table below,

- Rank the datapath designs in terms of achieved CPI.
- Rank the datapath designs in terms of design complexity.
- Rank the datapath designs in terms of chip area.

Use the mnemonics (SC, MC and P) in the table below.

	Largest	Middle	Smallest
CPI	MC	P	SC
Complexity	P	MC	SC
Area	P	SC	MC

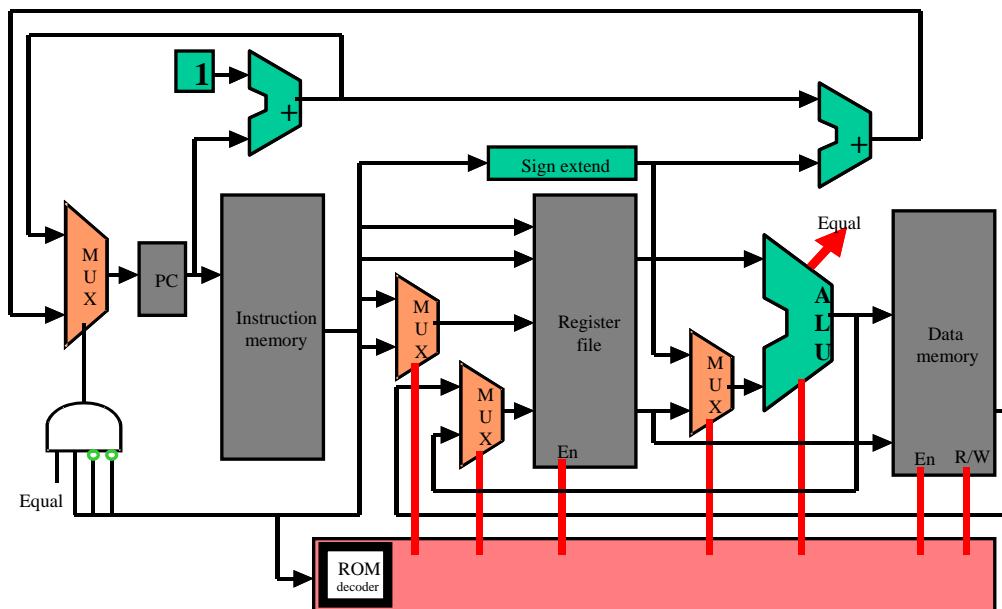
2. Single cycle datapath [10 points]

Consider the schematic of the single-cycle datapath for the LC2K processor that we studied in class. We want to extend our ISA and implement an additional instruction on this datapath. The new instruction is the **LWI** instruction. The instruction is an I-type instruction and it operates as follows:

LWI regA regB offset

regB = MEM[regA + offset]; regA = regA + 1;

that is, it reads a word at the address specified, and then it increments the base address register. This instruction is very useful when operating on arrays. Below, mark how we could modify the LC-2K datapath to accommodate this extra instruction. (Ignore the issues related to the opcode encoding limitation). Draw your changes on the diagram below, and also list them all below the diagram.



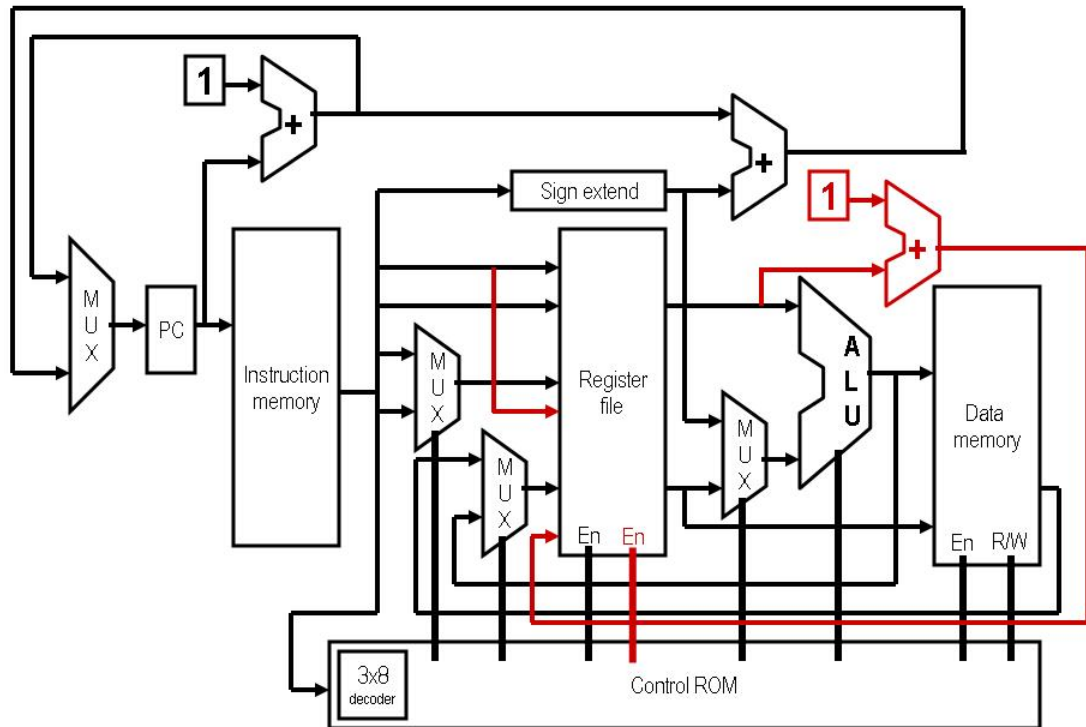
Changes list:

1. New or modified hardware blocks:

Answers on next page

2. Additional control signals:

3. Additional wire connections:



All modifications are in red on the above diagram . LWI performs the same operation as an LW instruction, but must perform the additional work of incrementing the register regA and storing the new value in regA.

New/modified hardware blocks:

1. Add an incrementer
2. Add a register file write port

Additional control signals

1. Write enable signal for the second write port

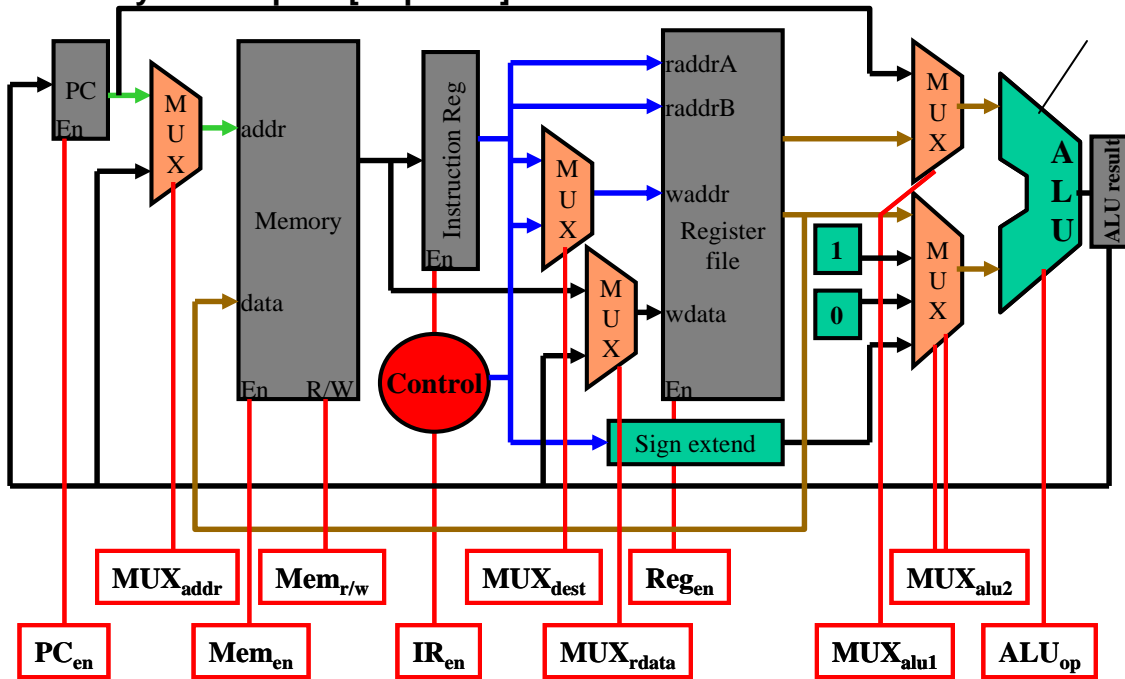
Additional wire connections

1. Connect the regA line from the register file to the input of the incrementer
2. Connect the output of the incrementer to the second RF write port – data signal
3. Connect the first RF read port address (which specifies the index of regA) to the address line of the second RF write port

Notes:

2. A few students devised a clever solution which avoids adding an incrementer. They implemented the incrementer using the adder that computes the new PC for BEQ instructions. The adder is not used during a LWI instruction, so it can be used as incrementer by just adding a couple of MUXes to its inputs.

3. Multicycle datapath [15 points]



(a) [5 pts] Consider the above LC-2K multi-cycle datapath covered in lecture shown above. Assume that the most significant select bit of **MUX_{alu2}** has a stuck-at-zero defect (that is, its value has been hardwired to zero, so that the multiplexor is always forced to select one of its two top entries). Circle all the LC-2K instructions below that can still execute correctly:

ADD BEQ JALR LW NAND NOOP SW

(b) [10 pts] For this part, consider the fully operational multi-cycle datapath above. Suppose that we want to provide support for a new instruction, **Mcopy**, which has the following semantic:

Mcopy regA regB offset $MEM[regB] = MEM[regA+offset]$

What additional hardware and control signals must be added to support the new instruction? [You can only add MUXes and wires] Draw your changes on the above datapath and also make sure to list them all below. How many cycles does the **Mcopy** instruction take to execute?

Changes list:

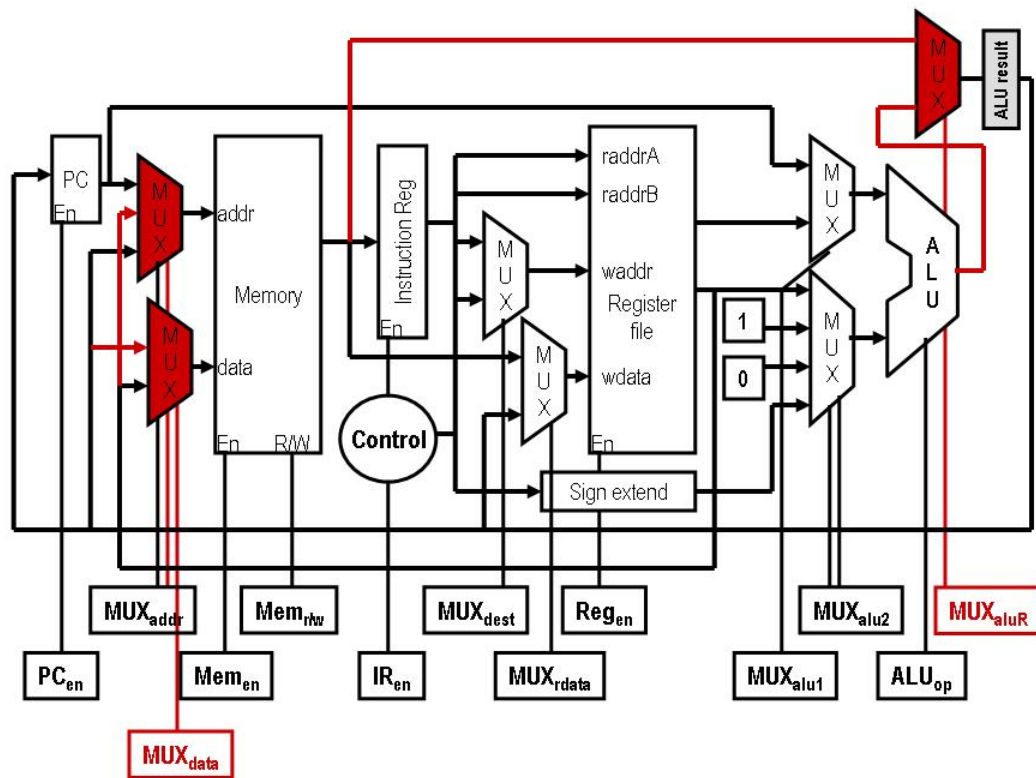
1. New or modified hardware blocks:

Answers on next page

2. Additional control signals:

3. Additional wire connections:

#cycles:



All modifications are in red on the above diagram – The Mcopy instruction takes 5 cycles, the same as the normal LW as the write to memory substitutes for the register write.

New/modified hardware blocks:

1. Memory address MUX widened to become 3-1
2. Add 2-1 MUX to memory data input
3. Add 2-1 MUX to ALU result latch

Additional control signals

1. Add 2nd control signal to memory address MUX
2. New 2-1 memory data MUX needs new control signal (MUXdata)
3. New 2-1 ALU result latch MUX needs new control signal (MUXaluR)

Additional wire connections

1. Connect regB to memory address (new middle input to memory addr MUX)
2. Connect ALUresult to memory data (new top input to memory data MUX)
3. Connect memory output to ALUresult (new top input to ALUresult MUX)

Notes:

1. Value read out from memory must be latched in order to write it back into the memory.
2. Another popular solution was to connect memory output to the ALUinput1 MUX, then add 0 to the value to store it into ALUresult. This results in a 6 cycle Mcopy.

4. Pipeline performance [9 points]

Consider the following three pipeline processor implementations. Assume that all 3 pipelines can achieve a CPI of 1, except for mis-predicted branches.

- **Implementation X:** A pipeline length of 20 stages with branches resolving in stage 7, running at 666MHz.
- **Implementation Y:** A pipeline length of 30 stages with branches resolving in stage 12, running at 1GHz.
- **Implementation Z:** A pipeline length of 25 stages with branches resolving in stage 5, running at 833MHz.

For all processors, the branch predictor is located in stage 1.

a) [1pt] What are the cycle times for each of these pipelines, in nanoseconds?

$$\text{Implementation X: } 1/(666 * 10^6) * 10^9 = 1.502 \text{ ns}$$

$$\text{Implementation Y: } 1/(1 * 10^9) * 10^9 = 1.000 \text{ ns}$$

$$\text{Implementation Z: } 1/(833 * 10^6) * 10^9 = 1.200 \text{ ns}$$

b) [3 pts] Now consider that you are the CTO at Intel and you want to choose a pipeline implementation for your next generation processor. You know that the branch predictor in your architecture can achieve 95% prediction accuracy. If you were to select a pipeline design purely based upon on minimum branch misprediction penalty in nanoseconds, which architecture would you choose? You must show your work clearly for full credit.

Branch misprediction penalty = #cycles/mispredict * cycle time

$$\text{Option X: } (7 - 1) \text{ cycles} * 1.502 \text{ ns/cycle} = 9 \text{ ns}$$

$$\text{Option Y : } (12 - 1) \text{ cycles} * 1.000 \text{ ns/cycle} = 11 \text{ ns}$$

$$\text{Option Z : } (5 - 1) \text{ cycles} * 1.200\text{ns/cycle} = 4.8 \text{ ns}$$

Implementation X: branch penalty **9 ns**

Implementation Y: branch penalty **11ns**

Implementation Z: branch penalty **4.8 ns**

Best Implementation (circle one): X Y Z

c) [5 pts] Now, consider pipeline Implementation X. Assume this is the only pipeline available to you, and as CTO you have two choices. The next generation of silicon technology has given you extra transistors to work with, and you can either use them to build a better branch predictor that will achieve 98% accuracy instead of 95%, OR to build extra hardware to allow branches to resolve in stage 5 instead of stage 7. Which approach will provide the best average CPI for a workload with 20% branches? Show all your calculations to receive credit.

$CPI = \text{Base CPI} + \text{Stall CPI due to mispredicts}$

Option 1: 98% accuracy, resolve stage 7, assume 100 instructions

$\text{Stall CPI} = (20 \text{ brs} * 0.02 \text{ mispredict rate} * (7 - 1) \text{ cycles per mispredict}) / 100 \text{ instrs}$
 $= .024$

$CPI = 1.0 + .024 = 1.024$

Option 2: 95% accuracy, resolve stage 5, assume 100 instructions

$\text{Stall CPI} = (20 \text{ brs} * 0.05 \text{ mispredict rate} * (5 - 1) \text{ cycles per mispredict}) / 100 \text{ instrs}$
 $= 0.04$

$CPI = 1.0 + .04 = 1.04$

Option 1 is better

Best choice (circle one):

Better prediction accuracy

Solve branches in stage 5

5. Pipeline datapath [13 points]

Imagine you want to add a new instruction “mul” to LC-2K7. Since “mul” is a long latency operation, you must split the EX stage of your 5 stage pipeline into 2 separate stages called EX1 and EX2, in order to maintain the short clock cycles of your current implementation.

a) [1 pt] Assuming we have no register-to-register forwarding and are using a WB/END pipeline register, how many stages ahead of EX1 must be checked for forwarding hazards?

4

b) [3 pt] List the additional stall scenario(s) that must be detected with this new pipeline (i.e. in addition to the lw-immediate use stall).

lw-use-use

mul-use (or r-type-use)

c) [9 pts] Assume that all original LC-2K R-type opcodes compute their result by the end of the EX1 stage, and can be forwarded at that time.

In the next page we prepared a program and a timing diagram for you. Fill in the timing diagram for the program. Indicate bubbles by X-ing out a stage, and indicate all required forwarding with arrows. The instruction column should indicate the instruction number and opcode being dealt with. You may not need to use the entire table. We have filled in the first row to get you started. KEEP IT NEAT. If we cannot read it, we cannot grade it.

5 - Continued.

Program:

- 1: lw 0 1 25
- 2: add 1 1 2
- 3: mul 1 2 3
- 4: nand 3 1 4
- 5: add 3 1 5

INST	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
1- lw	IF	ID	EX1	EX2	MEM	WB										
2-add		IF	X	X	ID	EX1	EX2	MEM	WB							
3- mul					IF	ID	EX1	EX2	MEM	WB						
4- nand						IF	X	ID	EX1	EX2	MEM	WB				
5-add								IF	ID	EX1	EX2	MEM	WB			

6. Branch predictors [6 points]

Consider a program with the following branch characteristics:

- 80% of the branches are taken
- 20% of the branches are not taken.

Of the taken branches,

- 30% are forward and
- 70% are backward.

The non-taken branches are

- 75% forward and
- 25% backward.

In addition,

- 80% of the backward-taken branches were taken also in their previous occurrence.
- However, none of the forward-taken branches were the taken in their previous occurrence.
- The non-taken branches are always taken the next time they are executed.

For each of the following branch prediction schemes, compute their accuracy rate (that is, what fraction of the time they predict the branch correctly):

[1 pt] Predict taken: 80% (the data is provided in the problem's text)

[2 pts] Predict backward taken, forward not taken:

The predictor predicts correctly at each occurrence of a backward taken branch and also of a forward not taken branch: $80\% * 70\% + 20\% * 75\% = 71\%$

[3 pts] Predict same as last time:

The branches that are the same as last time are only 80% of the backward taken. All other branches change taken/non-taken in the next occurrence.

So: $80\% * 70\% * 80\% = 44.8\%$

7. Caches [14 points]

Given a cache with the following configuration: total size is 8 bytes, block size is 2 bytes, 2-way associative, LRU replacement. The memory address size is 16 bits and is byte addressable.

For the following reference stream, indicate on the table below whether each reference is a hit or miss, and for misses specify the type (compulsory, conflict, capacity):

Addresses: 0, 1, 3, 5, 12, 1, 2, 9, 4

Addr	0	1	3	5	12	1	2	9	4
Hit or Miss	M	H	M	M	M	M	H	M	M
Compulsory	X		X	X	X			X	
Conflict						X			
Capacity									X

8. Cache performance [10 points]

On an architecture with the following specification:

- No data cache
- Perfect instruction cache (i.e., it never misses)
- Memory instructions take 5 cycles
- All other instructions take 1 cycle
- Perfect branch prediction (i.e., no stalls for branch mispredictions)

The CPI for a specific workload is not satisfactory. This workload has the following characteristics:

- Total number of instructions: 1500
- Number of memory instructions: 600
- Number of non-memory instructions: 900

We have two options to improve the performance on this workload:

Option 1: Hardware solution:

We add a data cache to the architecture. We know that the miss rate for the data cache is 20% for this workload. A data cache hit takes 1 cycle otherwise 6 total cycles (5 cycles for the miss and 1 cycle to transfer values to/from the register file).

Option 2: Compiler solution:

The compiler has a load-store elimination optimization that can eliminate 85% of the memory instructions and replace them with 2 arithmetic instructions.

Which of the two options improves the execution time the most? [Calculate the total number of cycles for each solution and show all of your work]

Calculations for Option 1

$$900 + 600 \times (.8 + 0.2 \times 6) = 2100 \text{ cycles}$$

Greater improvement

Calculations for Option 2

$$900 + 600 \times (.85 \times 2 + .15 \times 5) = 2370 \text{ cycles}$$

9. Hard problem on caches [10 points]

Consider two 8-byte caches with block size of 2 bytes, one is direct-mapped and one is fully associative. The replacement policy is LRU. Both have all lines marked invalid at the beginning of execution. Assume memory addresses are 8-bit and memory is byte addressable.

Is it possible to construct a string of addresses such that the direct-mapped cache will get a hit before the fully associative cache ? If so, find a possible shortest sequence, otherwise explain why not.

It is possible:

0 2 4 6 10 0