University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences
Computer Science Division

CS 61c J. Wawrzynek

Spring 2006

Machine Structures
Midterm III

ID Number: _____ Your Name: _____

Your TA: _____ Your login: _____

Left ID: _____ Seat No. _____ Right ID: _____

This is an open-book exam. You are allowed to use any books and notes that you wish. No calculators or electronic devices of any kind, please. You have 3 hours. Each question is marked with its number of points.

This exam booklet should have 23 printed pages. Check to make sure that you have all the pages. Fill out the information above **completely** and put your student ID neatly on each page.

Show your answers in the space provided for them. Write neatly and be well organized. If you need extra space to work out your answers, you may use the back of previous questions. However, only the answers appearing in the proper answer space will be graded.

Good luck!

problem	maximum	score
1	6pts	
2	8pts	
3	6pts	
4	10pts	
5	5pts	
6	14pts	
7	6pts	

problem	maximum	score
8	7pts	
9	3pts	
10	4pts	
11	11pts	
12	7pts	
13	3pts	
total	90pts	

1. [6 points] CPU Pipelining.

   Consider a MIPS processor with a pipeline structure different than the one discussed in class and the book. In this processor, the caches (instruction and data) and the ALU are pipelined to two stages each. Therefore the overall processor pipeline has the following eight stages:

   IF1, IF2, ID, EX1, EX2, DM1, DM2, WB

   While, the same mechanisms employed with the standard 5-stage MIPS pipeline are also employed on this new 8-stage pipeline (branch delay, ALU forwarding, load interlock), they may have limited effect, and therefore this pipeline may perform differently.

   Without reordering the instructions, write down the number of cycles needed to execute the following instruction sequences, including the time to drain the pipeline. The pipeline must be fully drained after each sequence.

   (a)
   ```
               addi $1, $0, 1
               addi $2, $0, 1
               addi $3, $0, 3
               addi $4, $0, 4
               beq  $1, $2, exit
               addi $5, $0, 5
               addi $6, $0, 6
               addi $7, $0, 7
   exit:       addi $8, $0, 8
   ```

   15 cycles

   2 points for correct answer

   -1 point for incorrect branch delay amount

   (b)
   ```
               addi $1, $0, 1
               addi $2, $0, 2
               addi $3, $1, $2
   ```

   11 cycles

   2 points for correct answer

   (c)
   ```
               addi $1, $0, 1
               addi $2, $0, 2
               lw   $3, 100($0)
               addi $4, $3, $1
   ```

   14 cycles

   2 points for correct answer

   -1 point for incorrect stall amount

2. [8 points] Caches I.

   (a) [1] Write down a formula for "average memory access time" for a system that has a *three* level cache hierarchy (L1, L2, and L3 caches). Ignore TLB access time.

   $hittime_{L1}+missrate_{L1}(hittime_{L2}+missrate_{L2}(hittime_{L3}+missrate_{L3}*misstime_{L3}))$

   1 point

   (b) [4] Consider a *direct-mapped cache* with 1 Byte per block and 4 blocks total. We plan to use it on a computer with 8-bit addresses and byte addressing. The cache is initially empty.

   The processor executes a sequence of load instructions from a list of memory addresses in the order listed in the table below. For each, indicate if the result is a hit or a miss, by writting hit or miss in the box. Also, for each miss, indicate the reason as one of conflict, or compulsory.

address (hex)	hit/miss	reason
00	miss	compusory
02	miss	compusory
01	miss	compusory
03	miss	compusory
04	miss	conflict
02	hit	
01	hit	
0B	miss	conflict

   0.25 points per hit/miss, 0.25 points per reason

(c) [1] Assume technolgy has advanced to a point that we can build a cache the same size as the physical memory. Assume also that we want to *guarantee* the highest possible cache hit rate in the system. Which of the following cache organization will result in the *lowest circuit complexity*, yet guaranteeing the highest possible hit rate?

A. Direct-Mapped

B. 2-Way Set Associative

C. 4-Way Set Associative

D. Fully Associative

A, 1 point

(d) [2] For the purpose of data cache addressing, two students proposed the following two ways of assigning the address fields:

Student A		
tag	index	offset
18	10	4

Student B		
index	tag	offset
10	18	4

With normal day-to-day mix of programs, which student's design is more likely to have a higher cache hit rate?

A. A has higher hit rate

B. B has higher hit rate

C. Design of A will cause cache malfunction

D. Design of B will cause cache malfunction

E. A and B are equally good.

A, 1 point

With an artificially generated random access pattern, which student's design is more likely to have a higher cache hit rate?

A. A has higher hit rate

B. B has higher hit rate

C. Design of A will cause cache malfunction

D. Design of B will cause cache malfunction

E. A and B are equally good.

E, 1 point

3. [6 points] Caches II.

Consider, the following is a program:

```
#define SIZE 64
int* a;
int main() {
    a = malloc(sizeof(int) * SIZE * SIZE);
    FillMatrix(a);
    FlushCache();
    return CountZero(a);
}
```

where `FillMatrix(a)` fills the array with values, and `FlushCache()` invalidates all entries in the cache. Now, consider the following 2 implementations of `CountZero()`:

```
    <<< Version A >>>            |     <<< Version B >>>
int CountZero(int* a) {          | int CountZero(int* a) {
  int row, col, z=0;             |   int row, col, z=0;
  for (row=0; row<SIZE; row++)   |   for (col=0; col<SIZE; col++)
    for (col=0; col<SIZE; col++) |     for (row=0; row<SIZE; row++)
      if (a[row * SIZE + col]==0)|       if (a[row * SIZE + col]==0)
        z += 1;                  |         z += 1;
  return z;                      |   return z;
}                                | }
```

(a) [2] If the code is run in a system with 1MB of direct-mapped cache with block size of 32 words which version of `CountZero()` will run faster?

A. Version A will be faster

B. Version B will be faster

C. They will take the same time to finish

C, 2 points

(b) [2] Given the same cache as above, if we change SIZE to 1024, which version of `CountZero()` will run faster?

A. Version A will be faster

B. Version B will be faster

C. They will take the same time to finish

A, 2 points

(c) [2] If the cache is now changed to 4-way set associative, with the remaining parameter same as above, and SIZE = 1024, which version of `CountZero()` will run faster?

A. Version A will be faster

B. Version B will be faster

C. They will take the same time to finish

A, 2 points

4. [10 points] Virtual Memory.

   Suppose we have a virtual memory system with the following parameters:

   - 16-bit word length
   - 16-bit virtual address
   - 4M bytes physical memory (M ≡ 1024*1024)
   - 512 byte page size
   - 8 entry TLB
   - A 8-bit wide process id (PID) register that stores the PID of the current running process

   (a) [2 points] The following resource limit is set for all programs running on this computer:

   - Fixed stack size of 8k bytes (k ≡ 1024) $2^{13}/2^9 = 2^4 = 16$
   - Fixed heap size of 8k bytes $2^{13}/2^9 = 2^4 = 16$
   - Maximum code size of 16k bytes $2^{14}/2^9 = 2^5 = 32$
   - Maximum static data size of 4k bytes $2^{12}/2^9 = 2^3 = 8$

   Assuming the page table size is fixed for all processes, how many entries are needed in the page table for each process? *2 points*

   72 entries

   (b) [2 points] Assume each entry of the page table has the following fields:

Name	purpose
OnDisk	1-bit field: 1 if this page is on harddisk, 0 otherwise
VPN	Virtual page number to be translated
PPN	Physical page number translated to

   What is the number of bits needed for the VPN field? *1 point*

$$16\ bit\ virtual\ addr - 9\ bits\ offset = 7\ bits$$

   What is the number of bits needed for the PPN field? *1 point*

$$4M = 2^{22} \rightarrow 22\ bit\ phys\ addr - 9\ bit\ offset = 13\ bits$$

(c) [5 points] The computer described on the previous page is currently running 2 programs, $P1$ and $P2$. The TLB is fully associative with a true LRU replacement policy. The table below shows the sequence of memory accesses generated by both $P1$ and $P2$. For each memory access indicate whether it generates a TLB hit (Hit), or miss (Miss). The TLB starts out empty.

0.5 points hit/miss, on 9 and 10, must be both misses or both hits

Time	Current Process	Memory Location (hex)	Hit/Miss
1	P1	F000	miss
2	P1	0200	miss
3	P1	010F	miss
4	P1	000C	hit
OS Swap in P2			
5	P2	F000	miss
6	P2	0404	miss
7	P2	030C	miss
8	P2	F10F	hit
OS Swap in P1			
9	P1	020F	miss or hit
10	P1	F004	miss or hit

(d) [1 points] When TLBs were first invented, they were often implemented with a *fully-associative* organization. In recent years, TLBs have been implemented with other cache organizations, such as *set-associative* or *direct-mapped*. Briefly, explain why this is so.

0.5 points for mentioning that TLBs are bigger nowadays

0.5 points for mentioning the benefits in terms of complexity, cost, or speed

5. [5 points] CPU Performance.

   Through the use of a set of benchmark programs we discover that a two instruction sequence (shift followed by add) is used many times in our programs. In fact, the two instruction sequence accounts for 20% of the total instructions executed. We decide to redesign our processor to include a single shift–add instruction that can be used in place of all the old two instruction sequences of shift followed by add. However, the new processor has a lower maximum clock frequency: $freq_{old}/freq_{new} = 1.1$.

   Which processor (old or new) has higher performance and by how much?

   *execution time = seconds/cycles * CPI * number of instructions*

   *old time = 1/(old freq) * CPI * previous number of instructions*

   *new time = 1/(new freq) * CPI * new number of instructions*

   new freq = old freq/1.1

   20% of instructions were reduced by half so that is 10%, so the new program has 90% of the original instructions.

   *new time = 1.1/(old freq) * CPI * new number of instructions * 0.9*

$$\frac{old\ time}{new\ time} = \frac{1/(old\ freq) * CPI * oldnumberofinstructions}{1.1/(new\ freq) * CPI * oldnumberofinstructions * 0.9}$$

   old/new = 1/1.1/0.9 = 1/0.99 = 1.0101 so the old time is slower and the new system is faster

   2 points for calculating new number of instructions/ old number of instructions

   1 point for recognizing that CPI is the same for the old and new processors

   *1 point for getting the time formula: 1/freq * CPI * number of instructions*

   1 point for getting solutoin correct

   -0.5 points for small math errors

   -0.5 for using the wrong frequency

6. [14 points] Disk Performance.

You are responsible for the storage subsystem design of a new computer. Your system runs only one application. This application has the following characteristics:

- Code size (including .text and .data section) is P bytes long.
- It allocates dynamic memory in the heap during run time. The heap grows as needed but never exceed 1M bytes.
- It is allocated a fixed stack size of 1 physical memory page.

Your computer system has the following characteristics:

- It has 1M bytes of physical memory (M ≡ 1024*1024)
- It has a page size of P bytes; No virtual memory system; No cache.
- Single cycle CPU, 100 MHz clock rate
- It takes 4 CPU cycles to load/store one byte to/from memory

You need to decide on which kind of storage should be used in this system. You are given 2 disk choices, D1, D2. The following table compares the 2 choices:

	D1	D2
Technology	Magnetic Hard Disk	USB Flash Drive
RPM	10000	N/A
Average Seek Time	8ms	N/A
Controller Overhead	4ms	1ms
Transfer Speed (bytes per seconds) Read (from disk to CPU) Write (from CPU to disk)	 $50000P$ $50000P$	 $125P$ $25P$

Both devices are block based (the block size is P). A coprocessor handles loading data from, and writing data to disk. Therefore, no CPU cycle is needed.

(a) [4] What is the time, T_{A1} and T_{A2}, to load the application from disk into the memory system using D1 and D2 respectively? Show your work.

Loading takes 1 page = 1 block

For D1:

$$\begin{aligned} T_{A1} &= \text{Controller Overhaed} + \text{Avg Seek Time} + \text{Avg Rotation Time} + \text{Xfer Time} \\ &= 4ms + 8ms + 0.5 * 60/10000 + \frac{P}{50000P} \\ &= 4ms + 8ms + 3ms + 20\mu s \\ &= 15.02ms \end{aligned}$$

For D2:

$$\begin{aligned} T_{A2} &= \text{Controller Overhead} + \text{Xfer Time} \\ &= 1ms + \frac{P}{125P} \\ &= 9ms \end{aligned}$$

4 points: correct answer

3 points: formula mostly correct except for 1 error, or if more than P bytes are loaded

2 points: incorrect formalt, but showed effort

(b) [6] Assume your application has a page fault rate of f during run-time. That is, out of all memory operations (`lw`,`sw`), f of them generates page fault. This page fault rate does not include the page fault generated during application loading. Your application has the following instruction mix:

Type	CPI	Usage
arithmetic/branch instructions	1	80%
memory read (`lw`) instructions	4	15%
memory read (`sw`) instructions	4	5%

Total number of instructions executed to finish the application is 10^8. Let,

T_{R1} = time needed to read 1 page of memory from D1
T_{R2} = time needed to read 1 page of memory from D2
T_{W1} = time needed to write 1 page of memory to D1
T_{W2} = time needed to write 1 page of memory to D2
T_{E1} = expected execution time of the application using a system with D1
T_{E2} = expected execution time of the application using a system with D2

Expected execution time is the time the application is expected to spend finishing all 10^8 instructions. Do not include the time to load your applications in your execution time calculation.

Express T_{E1} and T_{E2} in terms of f, T_{R1}, T_{W1}, and T_{R2}, T_{W2}. Clearly show your work.

Assume all pages are dirty, each page fault takes 1 page write and 1 page read.

$$\begin{aligned} T_{E1} &= num\ of\ insts * cycle\ time * avg\ CPI \\ &= 10^8 * 10^{-8} * 1 * 0.8 + (0.15 + 0.05)(4 + f * PageFaultPenalty) \\ &= 0.8 + 0.8 + 0.2f(T_{W1} + T_{R1})/Cycle\ time \\ &= 1.6 + 2 * 10^7 f(T_{W1} + T_{R1}) \end{aligned}$$

Similarly

$$\begin{aligned} T_{E2} &= 1.6 + 0.2f(T_{W2} + T_{R2})/Cycle\ time \\ &= 1.6 + 2 * 10^7 f(T_{W2} + T_{R2}) \end{aligned}$$

6 points: Perfect or almost perfect

5 points: Almost perfect, with 1 term missing

3 points: Major term missing but at least shows understanding of program execution time

2 points: Incomplete program execution formula, but shows some effort

The above answer is what we expected, but we also accept the following answer:

$$\begin{aligned} T_{E1} &= 1.6 + 2 * 10^7(0.15fT_{R1}) + 2 * 10^7(0.05fT_{W1}) \\ T_{E2} &= 1.6 + 2 * 10^7(0.15fT_{R2}) + 2 * 10^7(0.05fT_{W2}) \end{aligned}$$

(c) [2] If $f = 0.01\%$, what is the value of T_{E1} and T_{E2} in seconds?

$$\begin{aligned} T_{W1} &= T_{R1} = T_{A1} \\ T_{R2} &= T_{A2} \\ T_{W2} &= 1ms + \frac{P}{25P} = 41ms \\ T_{E1} &= 1.6 + 2 * 10^7(0.01\%)(2 * 15.02ms) \\ &= 1.6 + 2 * 10^3(2 * 15.02ms) \\ &= 61.68s \\ T_{E2} &= 1.6 + 2 * 10^7(0.01\%)(9ms + 41ms) \\ &= 101.6s \end{aligned}$$

2 points: Correct calculation of T_{R1}, T_{R2}, T_{W1} and T_{W2}, AND show substitution into equations from part (b)

1 points: show some effort

NOTE: No point is decducted for arithmetic errors.

(d) [1] If performance is your main concern, based on your result from Part C, which disk will you choose?

D1

(e) [1] Which of the following modification to the system will reduce T_{E2} to *at least half* of it's current value? Circle one of A,B,C,D or E.

A. Put a cache into the system that changes CPI of memory operations to 2

B. Double the CPU clock speed to 200 MHz

C. Double the amount of memory to 2M bytes

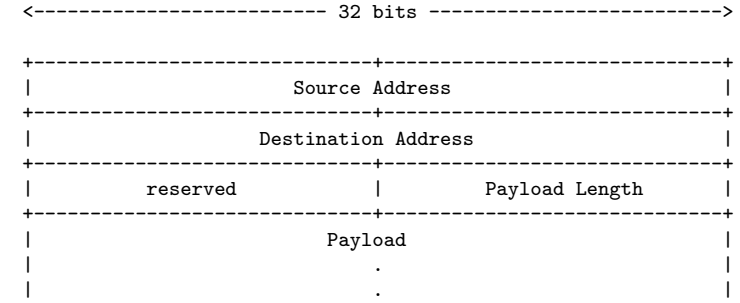
D. none of the above

E. all of the above

C

7. [6 points] C Programming and Networks.

   In the following problem, a *network packet* has the following structure:



   In the rest of this question we will use the term *body of the packet* to describe the payload *and* the 16 bits preceding it (its length).

   The program below should repeatedly read a packet from the network using `readPacket()`, check if the packet is meant for this computer, and invoke `processPacket()` on the body of the packet if it is.

```
1  /* gets a packet from the network and writes it into buf */
2  void readPacket(char* buf);
3
4  /* processes the body */
5  void processPacket(char* body);
6
7  char* readAndCheckPacket() {
8    char mydata[MAX_PACKET_SIZE];
9    int* destination;
10   readPacket(mydata);
11   destination = (int*)(&mydata);
12   destination += 4;
13   if (*destination != ADDRESS_OF_THIS_COMPUTER)
14    return NULL;
15
16   return (mydata+10);
17 }
18
19 void main() {
20   char* data;
21   while(1) {
22     data = readAndCheckPacket();
23     if (data != NULL)
24       processPacket(data);
25   }
26 }
```

This program contains three errors. Below, show how to fix each of the three errors by indicating the line containing the error and what the line should be changed to in order to fix the error.

line	revised line
8	char *mydata = (char *)malloc(MAX_PACKET_SIZE * sizeof(char));
11	destination = (int *)mydata;
12	destination += 1;

1 point for finding each line with errors

1 point for correcting the error correctly

8. [7 points] I/O and MIPS assembly.

   The following MIPS assembly program reads exactly 1024 keystrokes from the keyboard and sends them over a modem *in any order*. It also performs "other tasks" while waiting for data.

   The Keyboard
   The code at `intHandler` is the interrupt handler for the keyboard. When a key is pressed on the keyboard, it triggers the interrupt. The interrupt handler is then involked and should then read the character corresponding to the pressed key from address `0x00FF0000`.

   The Modem
   The modem's status can be read from address `0x00FA0000`; if this value is zero, the modem is ready to accept data. Data is sent to the modem by writing to address `0x00FFA0004`.

   The Buffer
   Since the order of the keystrokes does not need to be preserved, the programmer has chosen to store them in a *stack* at address `keyStack`. The word at address `stackTop` contains the address of the top of the stack.

```
1
2  stack:      .bytes 1024
3  stackTop:   .word  stack
4
5  intHandler:
6    la   $k0, 0x00FF0000     # $k0 = address of keyboard data
7    lb   $k0, 0($k0)         # $k0 = the keystroke
8    la   $k1, stackTop       # $k1 = address of stackTop variable
9    lw   $k1, 0($k1)         # $k1 = address of top of stack
10   sb   $k0, 0($k1)         # mem[stackTop] = newly-read keystroke
11   addi $k1, $k1, 1         # $k1 = new top of stack
12   la   $k0, stackTop       # $k0 = address of stackTop variable
13   sw   $k1, 0($k0)         # stackTop = $k1
14   eret
15
16 main:
17   la   $t0, 0x00FA0000     # $t0 = address of modem status
18   lw   $t1, 0($t0)         # $t1 = modem status
19   bne  $t1, $0, otherWork  # if modem not ready, do other stuff
20
21   la   $t0, stackTop       # $t0 = address of stackTop
22   lw   $t1, 0($t0)         # $t1 = address of top of stack
23   la   $t2, stack          # $t2 = address of bottom of stack
24   beq  $t2, $t1, otherWork # if stack top and bottom same, jump
25
26   addi $t1, $t1, -1        # $t1 = new top of stack
27   lb   $t3, 0($t1)         # $t3 = key at top of stack
28   sw   $t1, 0($t0)         # mem[stackTop] = new top of stack
29   la   $t0, 0x00FA0004     # $t0 = address of modem output
30   sb   $t3, 0($t0)         # modem output = $t3
31
32 otherWork:
33   # ....
34   j main
35
```

True/False, [0.5 points each]

T / F The keyboard input routine uses *polling*

F

T / F The keyboard input routine uses *memory-mapped I/O*

T

T / F The keyboard input routine uses *interrupt-driven I/O*

T

T / F The modem output routine uses *polling*

T

T / F The modem output routine uses *memory-mapped I/O*

T

T / F The modem output routine uses *interrupt-driven I/O*

F

[2 points]
This program has a *race condition*. If the keyboard interrupt is fired during a certain part of the `main` routine, the program will malfunction. Fill in the following two blanks:

> The program will malfunction if the interrupt occurs *after* the execution of line [ ] but *before* the execution of line [ ].

between line 22 and line 28

1 point for each line number

[1 point] When the program malfunctions, it will (choose one):

- Fail to transfer one of the keystrokes to the modem.
- Transfer one of the keystrokes to the modem twice.
- Get stuck in an infinite loop.
- Crash due to an invalid memory access.
- Output π to an accuracy of 12 digits.

Fail to transfer one of the keystrokes to the modem.

[1 point]

T / F Because the next key pressed is stored at address `0x00FF000A`, the computer must have at least $\texttt{FF000A}_{\text{hex}}$ bytes of memory in order to operate correctly.

F

9. [3 points] String Representation.

   In contrast to the ANSI C standard, early Macintosh computers stored strings as a *length byte* followed by the bytes of the string, with no terminator. This style of string is called *Pascal-style strings*, since they were first used in the Pascal programming language.

   [1 point]
   Are there any advantages to using Pascal strings instead of C strings? If so, name *one*.

   Yes, strings can contain the null character or strlen() runs in constant time

   [1 point]
   Are there any advantages to using C strings over Pascal strings? If so, name *one*.

   Yes, strings can be longer than $2^8 - 1$ *chars*

   [1 point]
   Would either of your answers above change if Pascal strings used an *int* (assumed to be four bytes long) rather than a *byte*? If so, how?

   Yes, strings can now be up to $2^{32} - 1$ *chars long, but every string requires an extra 3 bytes of space*

10. [4 points] MAL and TAL.

    [1 point]
    The MAL (MIPS Assembly Language) instruction `nop` can be translated into TAL (True Assembly Language) in many different ways. Give two possible translations which have different opcodes or function fields.

    slt $0, $0, $0

    add $0, $0, $0

    0.5 points each

    [3 points]
    Imagine that the assembler writer wants to add a new MAL instruction called `lwa`, *Load Word from Address*, which takes two arguments:

    ```
    lwa $rd, addr
    ```

    The first argument is a destination register. The second argument to the instruction is a 32-bit compile-time constant, just like the second argument to `la`. The instruction should load the 32-bit value *at the memory location contained in* `addr` and place this value in `$rd`.

    Show how the assembler should translate this MAL instruction into TAL. Your result should be a template containing `$rd` and `addr` somewhere, to be used by the assembler to generate one or more TAL instructions. *What remains should be entirely TAL instructions.*

    lui $at, upper(addr)

    ori $at, $at, lower(addr)

    lw $rd, 0($at)

    can substitute $rd for $at

    -1 for usng t/s/etc. registers

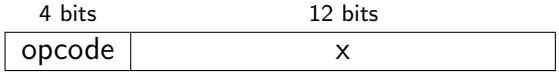
    -1 for using non-TAL instructions

    -1 for not dividing into upper and lower halves

    -1 for oring with $0 instead of $at, or not specifying ori operand

11. [11 points] X-processor Revisited.

This question is based on the X-processor from midterm exam 2. The X-processor is a 16-bit *accumulator-based* architecture. Instead of a general purpose register file, it has a single register, called the *accumulator* (ACC). All instructions use the same instruction format, shown below:

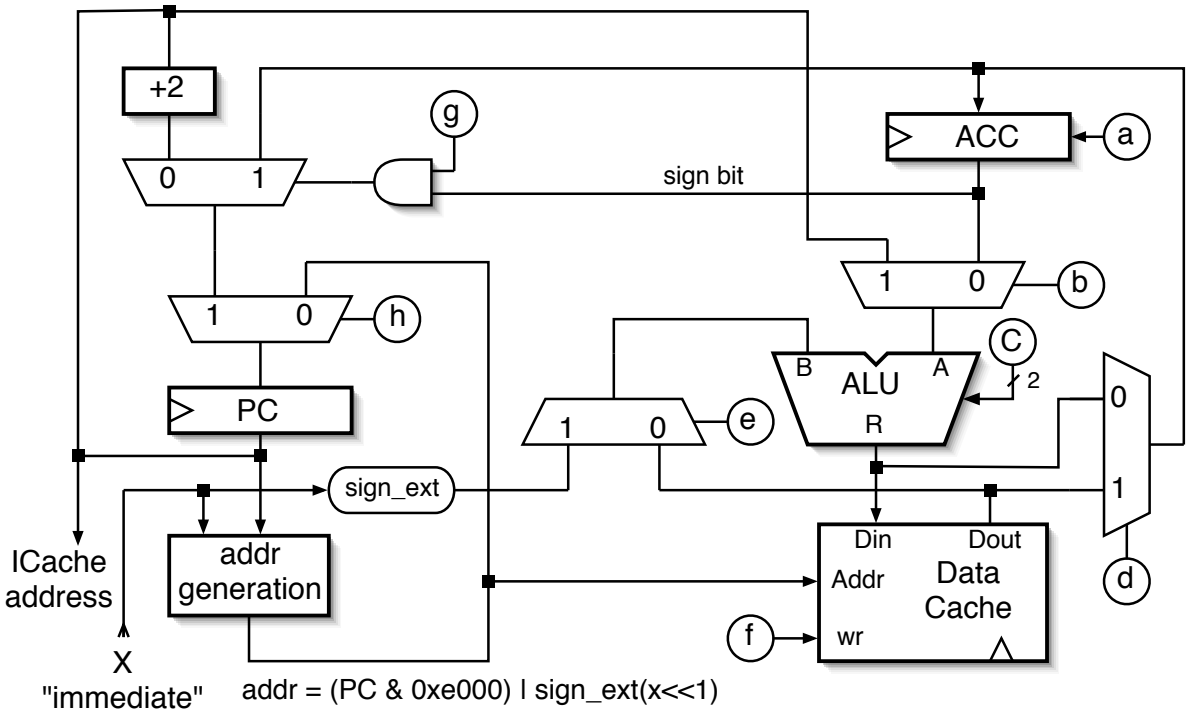


The "x" field is used by each instruction to help specify a memory address, or as an immediate operand. We would like to add two new types of memory load instructions, listed below:

instruction	**name**	**operation**
load word register	`lwr`	acc ←MEM[acc + sign_ext(x)]
load word memory	`lwm`	acc ←MEM[MEM[acc]]

The old datapath for X-processor is shown in the next page. The control inputs are labeled with the alphabet, indicated by circles on the diagram. The control signals are described below:

signal	**use**	**meaning**
a	ACC write enable	
b	multiplexor control	
C	alu_control	00: R=A+B, 01: R=A-B, 10: R=A NOR B, 11: R=A
d	multiplexor control	
e	multiplexor control	
f	data memory write enable	
g	branch	set to 1 on branch instruction
h	multiplexor control	



(a) Proper single-cycle execution of the `lwr` instruction requires a modification to the datapath and controller. As neatly as possible, make the simplest change to the datapath to allow execution of the `lwr` instruction. Briefly explain your change here.

Mux output of ALU with the addr line into the Date Cache. Call mux selector signal "i" and addr is the 0 input.

2 points for correct mux placement

Fill in the table below with the value of each control signal to properly execute the `lwr` instruction. If you needed to add new control signal(s) above, add column(s) to the table and write in value(s).

2 points for all the correct signals

a	b	C	d	e	f	g	h	i
1	0	00	1	1	0	0	1	1

(b) The `lwm` instruction will take more than one clock cycle to execute. Assuming the controller is changed appropriately, what change needs to be made to the datapath to allow multi-cycle instruction execution? Make the change on the datapath diagram and briefly explain it here.

Add a write enable signal for the PC register. Call this signal "j".

3 points for the write enable signal

Assuming the datapath has been modified as needed for multi-cycle execution, fill in the table below with the setting of the control signals for as many cycles as needed to execute the `lwm` instruction. Add new column(s) to the table as needed to accomodate new control signal(s) from part (a) above, and for multi-cycle execution.

2 points for all the correct signals

cycle no.	a	b	C	d	e	f	g	h	i	j
1	1	0	11	1	X	0	0	X	1	0
2	1	0	11	1	X	0	0	1	1	1

(c) Of course, the behavior of the `lwm` instruction could be emulated by a sequence of *single-cycle* instructions. Below list one advantage for each approach:

Name *one* advantage to multiple-instruction approach:

1 point

fewer datapath changes, less control complexity

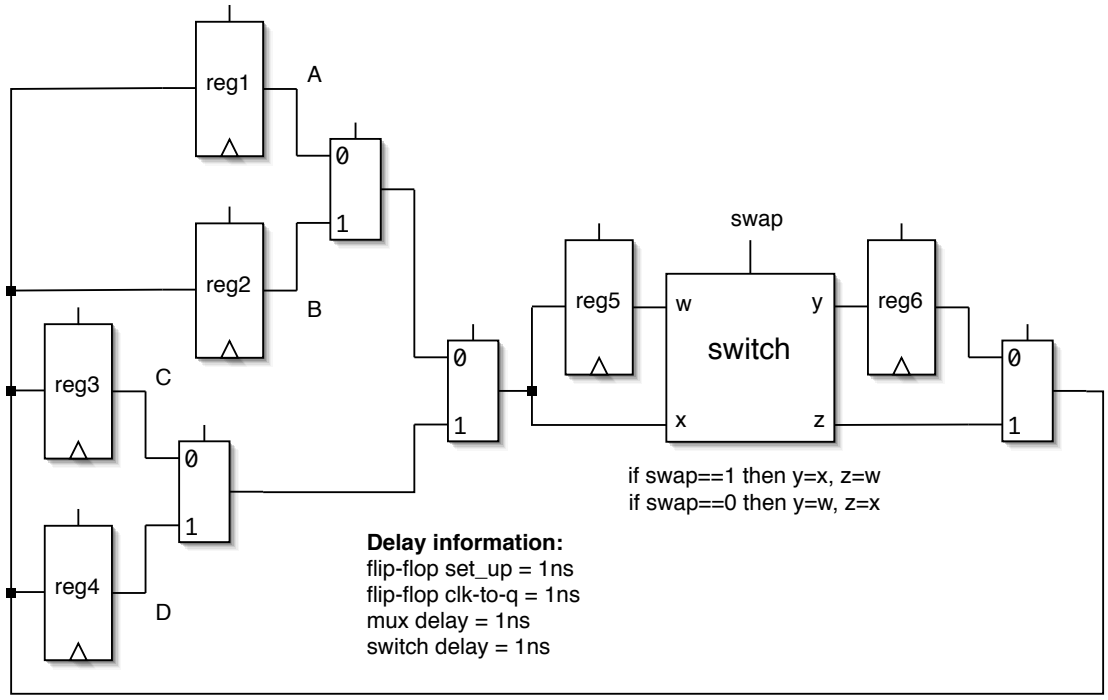
Name *one* advantage of new multiple-cycle instruction approach:

1 point

smaller code size

12. [7 points] Synchronous Digital Systems The diagram below shows a circuit comprising registers, multiplexors, and a combinational logic block labelled "switch". The switch passes its inputs straight through or crosses them as described below.

    Initially, reg1, reg2, reg3, and reg4 contain the values A, B, C, and D, respectively.



(a) Assuming the presence of a controller to set the control signals appropriately, what is the minimum number of cycles needed to reverse the set of values in registers 1–4. In other words, how many cycles will it take to get to a final state with reg1, reg2, reg3, and reg4 containing the values D, C, B, and A, respectively?

cycle: register transfer

1: $reg5 \leftarrow reg1$

2: $reg6 \leftarrow reg4, reg4 \leftarrow reg5$

3: $reg1 \leftarrow reg6, reg5 \leftarrow reg2$

4: $reg6 \leftarrow reg3, reg3 \leftarrow reg5$

5: $reg2 \leftarrow reg6$

scoring: 3 points for 4 cycles

5 points for 5 cycles

4 points for 6 cycles

3 points for 7 cycles

0 points for anything else

(b) Based on the delays listed in the diagram, how long (in ns) will it take?

$$5\ cycles * worst\ path\ delay$$

$$worst\ path\ delay = clk - to - q + mux + mux + switch + setup = 5ns$$

25ns total

Credit given if cycle count is consistent from part a AND worst path delay is correct for the path with that cycle count

13. [3 points] x86 Architecture.

(a) The original Intel x86 ISA was designed at a time when memory and registers were expensive. The same design constraints, of very limited hardware resources, applies to modern embedded systems. Embedded systems must meet the following constraints:

- Small hardware size
- Small amount of memory
- Low hardware complexity to reduce power consumption
- Moderate computing performance

However, most embedded system in real life use processors such as MIPS or ARM. Which of the following are possible reasons:

A The design of the MIPS (and ARM) ISA is inherently slow, and therefore cannot deliver the performance required for desktop applications.
B The Intel ISA is too complicated for simple hardware implementations as needed in embedded systems.
C The register-memory architecture of x86 ISA requires a lot of memory to run which is not available.
D All of the above.
E None of the above.

B

1 point

(b) Apple computer have recently started using Intel processors in place of PowerPC processors in all of their desktop and laptop computers. For each of the following, decide if it can be run on these new Intel-based Apple computers ***natively***.

Original Mac operating system OSX compiled for PowerPC based desktop.	*No*
Windows operating system compiled for your home PC.	*Yes*
An database system developed in 1990 for a 80486 processor.	*Yes*
A Java program compiled as byte code from your home PC	*No*

0.5 points per question