

EXAM 2

Tuesday, November 21, 2006

Instructions

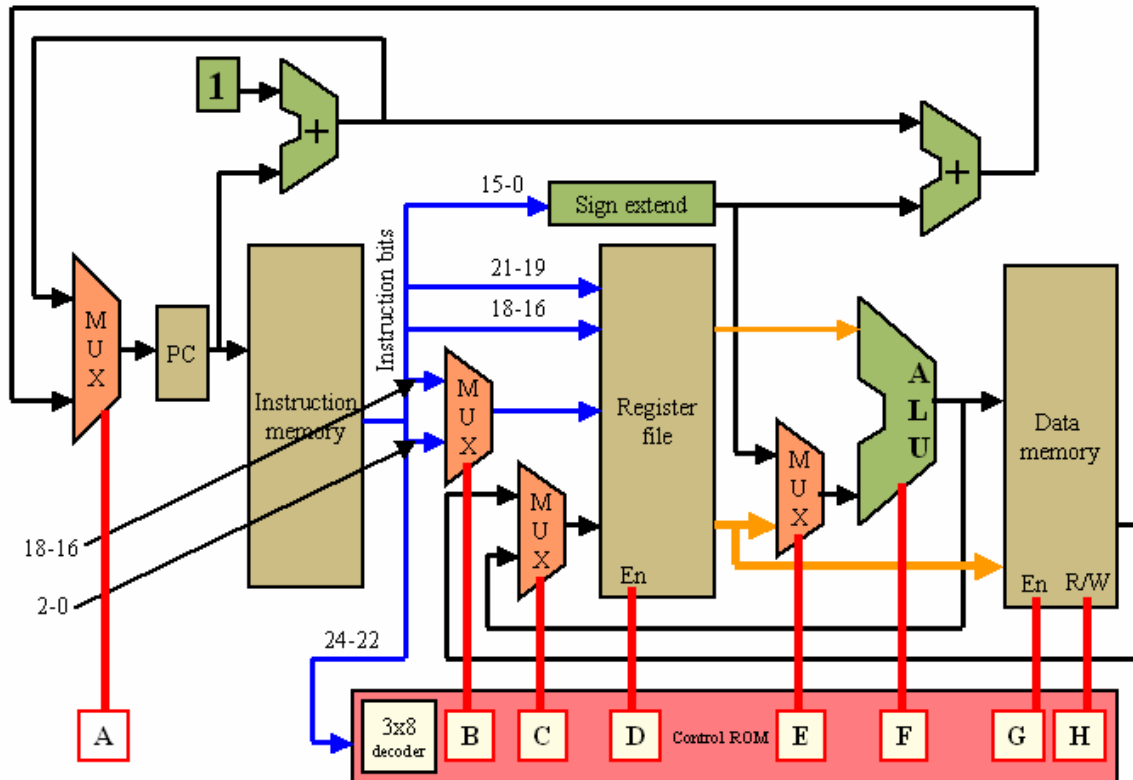
1. This is an open books/notes exam.
2. It comprises:
 - 3 short problems (20 points total).
 - 3 not-as-short problems (80 points total).
3. For each question, indicate your answer directly on the exam.
4. You have 80 minutes to complete the exam.
5. Please make sure you enter your name and UMID in the space below AND on the answer sheet.
6. By signing below, you certify that your conduct throughout the exam has been in accordance with the College of Engineering Honor Code.
7. At the end of the exam, please turn in this exam booklet.

Name _____

UMID _____

Signature _____

Section I: Short Answers [20 points]



1. [4 points] Assume you want to extend the LC2K6 single-cycle datapath (shown above) to support a jump instruction J , where “ J regA offset” results in a branch to the target address $\text{regA} + \text{offset}$. Among the 8 control signals A—H, pick all the signals that are safe to be set to “don’t care” for such an instruction.

- BCGH**
- BCDEFGH
- DGH
- BCE
- ABCDEFGH

2. [8 points total for parts (a) and (b)] Consider running the following assembly program on a 5-stage pipelined LC2K6 datapath similar to the one covered in lecture. Assume that on `beq`'s the pipeline stalls as appropriate until the PC is updated at the end of the EX stage. Also, assume that the pipeline supports a delayed branch instruction `dbeq` with a single delay slot.

```

FIESTA  lw    0    1    UM
        beq   0    1    UF
        add   1    2    2
UF      lw    0    3    USC
        beq   0    3    BCS
        add   3    4    4
BCS     lw    4    5    OSU
        add   5    6    6
        # return to caller function

```

- (a) [4 points] Consider replacing `beq` by `dbeq` to reduce execution time. How would you re-order the instructions (while preserving correctness) to make the most of the delayed branch feature?

```

FIESTA  lw    0    1    UM
        dbeq  0    1    UF
        lw    0    3    USC
        add   1    2    2
UF      beq   0    3    BCS
        add   3    4    4
BCS     lw    4    5    OSU
        add   5    6    6
        # return to caller function

```

- (b) [4 points] For your solution in (a), how many cycles will you save compared to running on the pipeline without delayed branching each time you call the function FIESTA? Assume that all labels point to valid locations in memory.

1 cycle from dbeq
1 cycle from eliminating load hazard
= 2 cycles total

1. [8 points for parts (a) and (b)] Consider a cache with the following configuration:

4-way set-associative
Block size = 4 bytes
32 cache lines
 2^{16} addresses in memory
Byte-addressable
LRU replacement policy

(a) [4 points] How many bits are in the tag?

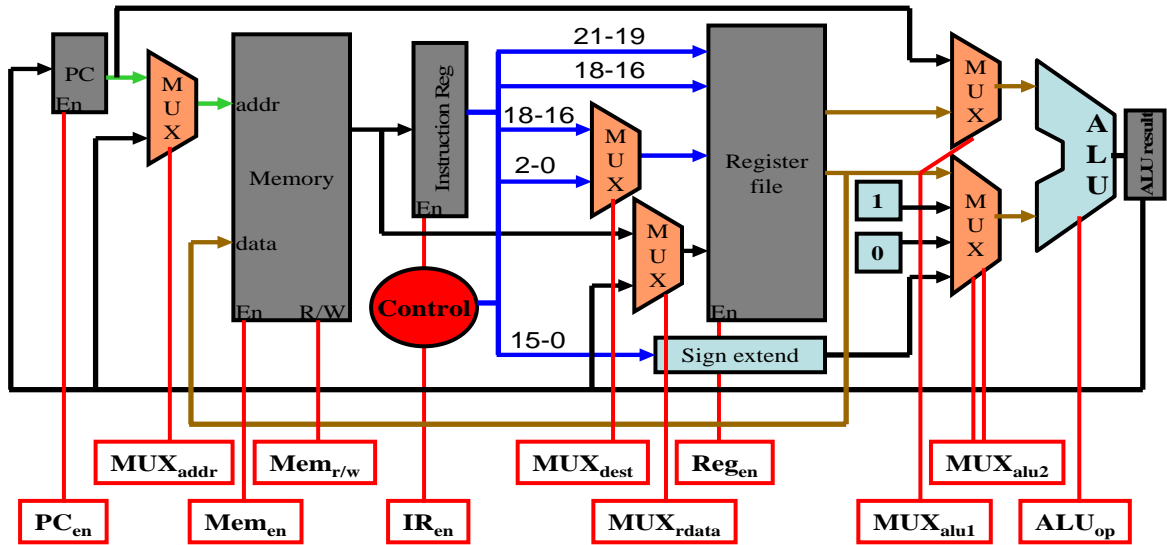
11

(b) [4 points] How many bits are in the set index?

3

Section II: Not-as-Short Questions [80 points]

Part A. Multi-Cycle Datapath [20 points]



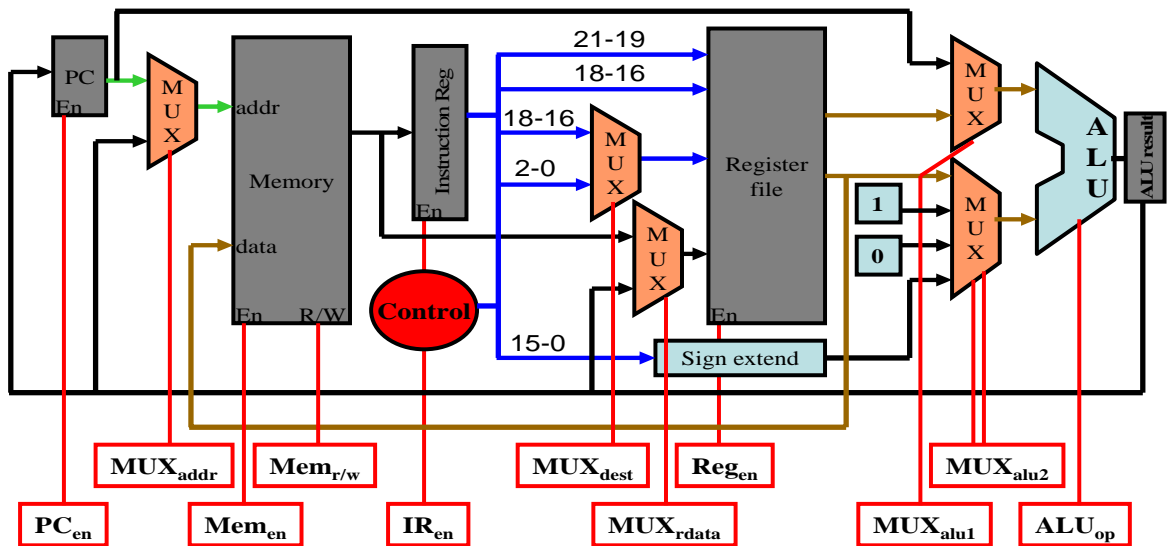
Consider the LC2K6 multi-cycle datapath covered in lecture (given above). We want to provide hardware support for a new R-type instruction:

$$destReg \leftarrow regA + (regA * regB)$$

This instruction adds the product of registers `regA` and `regB` to register `regA` and places the sum in register `destReg`.

1. [8 points] List any new hardware components and/or modifications to existing hardware components that are required to support the new LC2K6 instruction. Also, indicate your additions and/or modifications on the datapath in the previous page. Make sure you include all additions and/or modifications related to MUXes, functional units, registers, and datapath connections (i.e., wires).

Just in case you need it, here is another copy of the datapath from the previous page.



- **Augment ALU to perform multiply**
 - Requires extra control line to select correct operation
- **Wire from ALUresult to MUX_{ALU2}**
- **Enlarge MUX_{ALU2} to accommodate ALUresult**
 - Requires extra control line to select between 5 inputs

2. [12 points] Give a cycle-by-cycle description of the LC2K6 operation when executing the new instruction. For each cycle, give the following information:
- Single-sentence description of what the cycle is about.
 - Register updates.

Use as few cycles as possible. (Note that you may not need to use all the boxes provided.) To get you going, we provide the first 2 cycles.

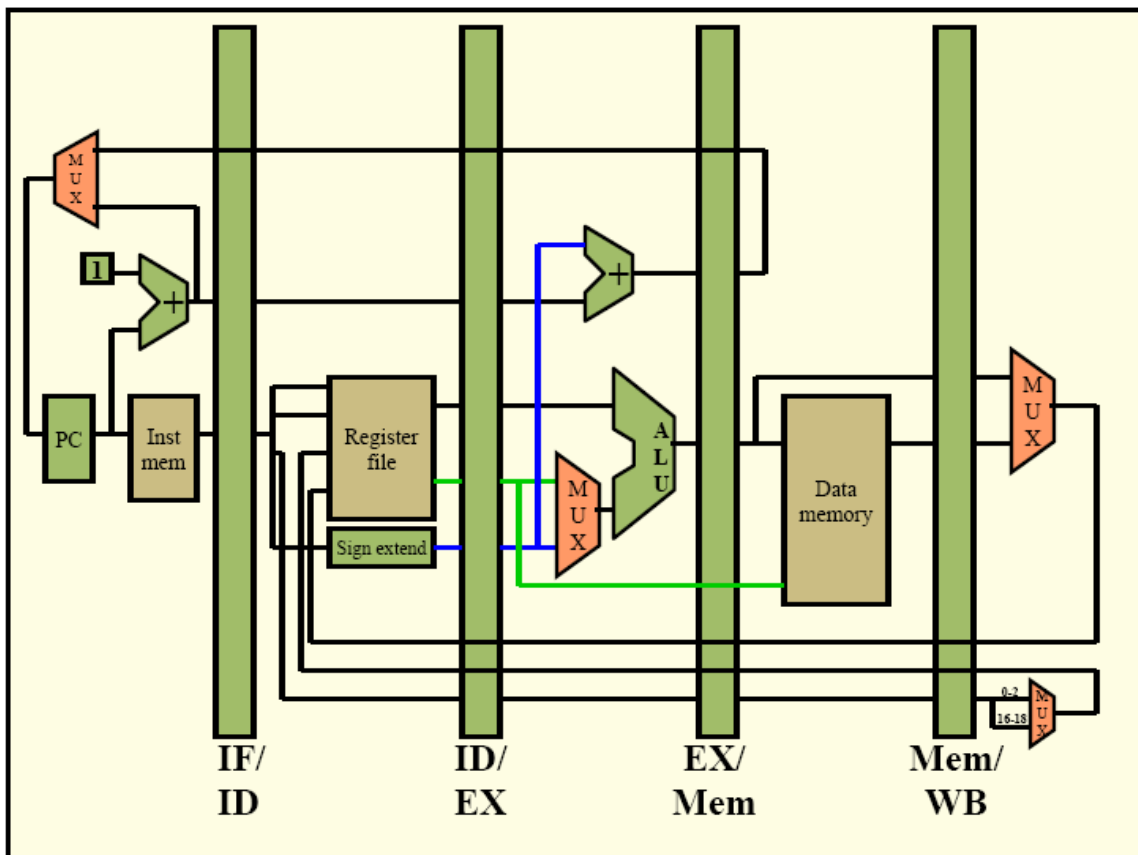
Cycle 1	Fetch instruction. Instruction Register \leftarrow new instruction ALU Result \leftarrow PC + 1
Cycle 2	Decode instruction and read registers. PC \leftarrow ALU Result
Cycle 3	Perform multiply. ALUresult \leftarrow regA * regB
Cycle 4	Perform addition. ALUresult \leftarrow ALUresult + regA
Cycle 5	Write back to register file. RegFile[destReg] \leftarrow ALUresult

Part B. Pipelining [44 points]

Suppose we want to add the following instruction to our ISA in order to support array manipulations.

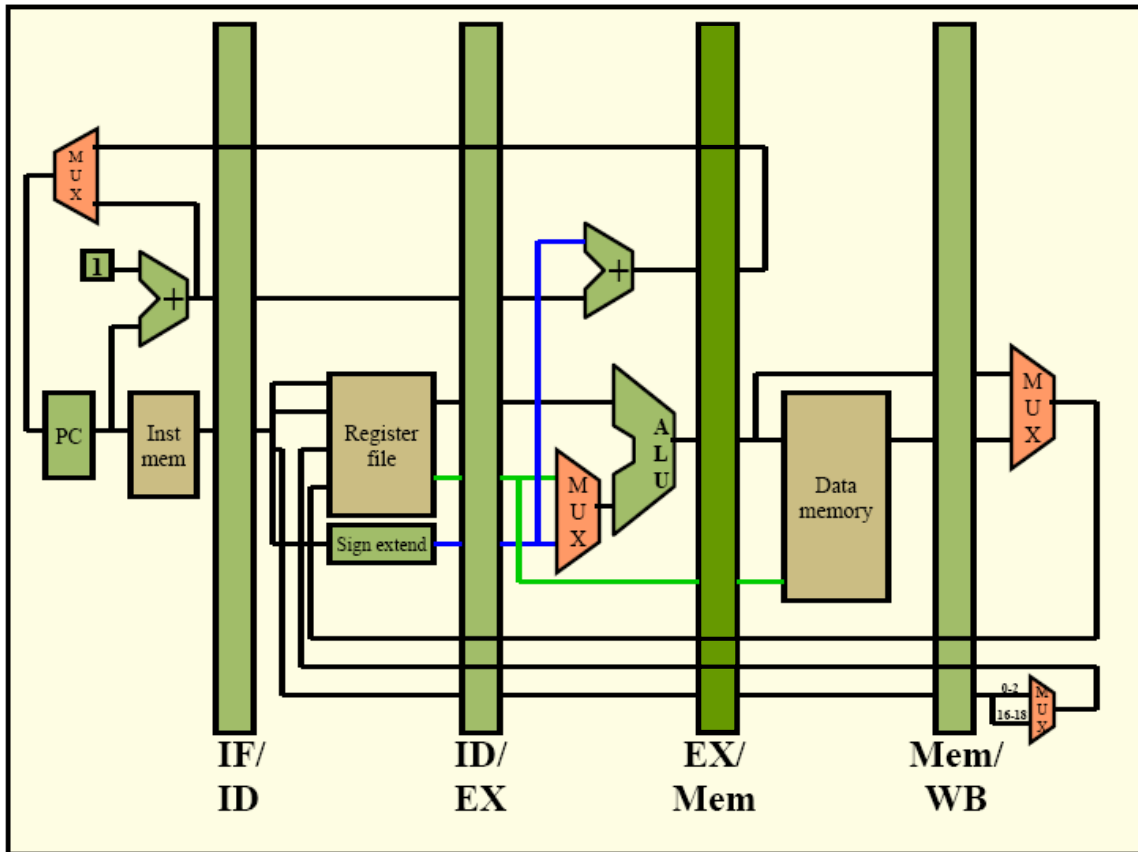
```
memcpy $r1 $r2 offset
```

The array value at memory address $\$r1 + \text{offset}$ is copied into memory address $\$r2$, and $\$r1$ is incremented to point to the next array element. The basic LC2K6 pipeline is shown below. Register $\$r1$ is read from the “upper” port of the register file, whereas $\$r2$ is read from the “lower” port.



1. [16 points] List any new hardware components and/or modifications to existing hardware components that are required to support `memcpy`. Indicate your additions and/or modifications on the datapath in the previous page. Make sure you include all additions and/or modifications related to MUXes, functional units, registers, and datapath connections. *Do not add new memories or new read/write memory ports to existing memory.*

Just in case you need it, here is another copy of the datapath from the previous page.



- **Extra adder in EX stage to increment \$ri**
- **Extra field in EX/MEM and MEM/WB pipeline registers to store \$ri++**
 - **Augment writeBack MUX to accomodate MEM/WB.ri++**
 - **Augment destMUX to accommodate bits 21:19 of instruction (ri)**
- **Extra MUX at data input of Data Memory**
 - **selects between EX/MEM.rd and MEM/WB.MemData**
 - **requires corresponding wire**
- **Extra mUX at address input of Data Memory**
 - **selects between EX/MEM.aluResult and EX/MEM.rd**
 - **requires corresponding wire**

2. [16 points] Give a cycle-by-cycle description of the LC2K6 operations **related to the execution of memcpy**. For each cycle, give the following information:

- Relevant stage(s) involved in execution of instruction (may be more than one).
- Short description of what the stage is about.
- Register updates.

Use as few cycles as possible. (You may not need all boxes provided.) To get you going, we provide the first 2 cycles.

Cycle 1	IF stage: Fetch memcpy. IF / ID \leftarrow memcpy instruction IF / ID \leftarrow PC + 1
Cycle 2	ID stage: Decode memcpy, read registers. ID / EX \leftarrow \$ri (reg A) ID / EX \leftarrow \$rd (reg B) ID / EX \leftarrow offset ID / EX \leftarrow PC + 1
Cycle 3	EX stage: Calculate \$ri + offset, \$ri+1 EX / MEM \leftarrow \$ri + offset EX / MEM \leftarrow \$ri + 1 EX / MEM \leftarrow \$rd EX / MEM \leftarrow PC + 1
Cycle 4	MEM stage: Read Mem[\$ri + offset] MEM / WB \leftarrow Mem[\$ri + offset] MEM / WB \leftarrow EX/MEM.ri++
Cycle 5	MEM stage: Write Mem[\$rd] and WB RegFile[\$ri] \leftarrow MEM/WB.ri++ (\$ri + 1) Mem[\$ri] \leftarrow MEM/WB.MemData (data from Mem[\$ri+offset])

3. [4 points] How many cycles is the latency of memcpv?

5

4. [8 points] Describe all cases, if any, in which memcpv may result in a pipeline stall.

Pipe always stalls for one cycle minimum (disable PC, ID/EX, EX/MEM, IF/ID). If the pipeline registers are augmented with additional fields it is possible to avoid stalling when memcpv is followed by R-type instructions.

If the instruction following memcpv uses \$ri then you must stall until WB (or introduce additional forwarding path to EX)

Part C. Cache Configuration [16 points]

Consider the following cache.

2-way set-associative
Block size = 4 bytes
Cache size = 32 bytes of data
 2^{16} addresses in memory
Byte-addressable
LRU replacement policy
Write-back
Write-allocate

For such a cache, 2 bits are used for the block offset, 2 bits are used for the set index, and 12 bits are used for the tag.

Consider the following trace of Read / Write accesses to individual bytes in memory (all cache lines are initially invalid).

address	0	1	5	23	0	39	5	100
type	R	R	W	R	R	W	R	W
H / M	M	H	M	M	H	M	M	M

1. [8 points] For each access, indicate whether it is a Hit or a Miss.
2. [8 points] What is the total number of bytes written back to the main memory?

8