Department of Electrical and Computer Engineering
The University of Texas at Austin
EE 460N/382N.1, Spring 2018
Instructor: Dr. Mattan Erez
TAs: Steven Flolid, Armand Behroozi, Abraham Gonzalez

Problem Set 1 Solutions

Question 1

A small section of byte-addressable memory is given below.

Address	Data
x5000	x21
x5001	x32
x5002	xB3
x5003	x03
x5004	x0C
x5005	x11
x5006	x6E
x5007	xA2

Add the 16-bit 2's complement numbers specified by addresses x5000 and x5006 if

a. The ISA specifies a little-endian format
   m[x5000/1] + m[x5006/7] = x3221 + xA26E = xD48F
b. The ISA specifies a big-endian format
   m[x5000/1] + m[x5006/7] = x2132 + x6EA2 = x8FD4

Question 2

A processor is used to primarily run a critical graphics benchmark. Floating-point additions (FPADD instruction) are responsible for 30% of the execution time of this benchmark. One proposal is to enhance the FPADD hardware and speed up this

operation by a factor of 8. The other alternative is to make all FP instructions in the graphics processor run faster by a factor of 2. FP instructions are responsible for half of the execution time of the benchmark. The design team estimates both alternatives to cost the same. Which design will you recommend? Explain.

1. Enhance FPADD(30% of execution time) by factor of 8
   $S_1 = \frac{1}{(1-fraction_{enhanced}) + fraction_{enhanced}/speedup_{enhanced}} = \frac{1}{(1-0.30)+(0.30/8)} = 1.356$

2. Enhance all FP(50% of execution time) instructions by a factor of 2
   $S_2 = \frac{1}{(1-fraction_{enhanced}) + fraction_{enhanced}/speedup_{enhanced}} = \frac{1}{(1-0.5)+(0.5/2)} = 1.33$

Hence, making the FPADD instructions run faster by the factor of 8 is recommended.

Question 3

The ISA is the contract between the hardware and the software.
The microarchitecture is a particular implementation of an ISA.
Compiler does not need to know any information about the microarchitecture to compile correctly.

Classify the following attributes of LC-3b as either a property of its microarchitecture or ISA:

a. There is no subtract instruction in LC-3b. (ISA)
b. The ALU of LC-3b does not have a subtract unit. (uarch)
c. LC-3b has three condition code bits (n, z, and p). (ISA)
d. The n, z, and p bits are stored in three 1-bit registers. (uarch)
e. A 5-bit immediate can be specified in an ADD instruction (ISA)
f. It takes *n* cycles to execute an ADD instruction. (uarch/ISA)
g. There are 8 general purpose registers used by operate, data movement and control instructions. (ISA)
h. The registers MDR (Memory Data Register) and MAR (Memory Address Register) are used for Loads and Stores. (uarch)
i. A 2-to-1 mux feeds one of the inputs to ALU. (uarch)
j. The register file has one input and two output ports. (uarch)

Question 4
Tradeoffs

a. What is the key advantage of the architecture–microarchitecture separation (20 words or fewer)?
   Software can be developed without thinking about hardware implementation. Likewise, hardware can evolve independently
b. What is an alternative to a microcoded microarchitecture (20 words or fewer)?
   Dedicated resources for each ISA instruction
   Store the next state of each combination of state and input.
c. What is a major advantage of choosing a microcoded microarchitecture (20 words or fewer)? Fewer bits to store in control unit / more efficient (in # gates) / easier to change
d. What is a major disadvantage of choosing a microcoded microarchitecture (20 words or fewer)? Additional microsequencer hardware is needed to determine next state / possibly slower (more ucode inst per ISA inst)
e. What is one disadvantage of condition codes (cc) (20 words or fewer)? Additional hardware is needed (cc register(s)) / branch instruction must be right after the instr. that sets cc (reduced parallelism) / have to compute cc every time even if not used for BR

Question 5

A zero-address machine is a stack-based machine where all operations are done using values stored on the operand stack. For this problem, you may assume that its ISA allows the following operations:

- PUSH M - pushes the value stored at memory location M onto the operand stack.
- POP M - pops the operand stack and stores the value into memory location M.
- OP - Pops two values off the operand stack, performs the binary operation OP on the two values, and pushes the result back onto the operand stack.

Note: To compute A - B with a stack machine, the following sequence of operations are necessary: PUSH A, PUSH B, SUB. After execution of SUB, A and B would no longer be on the stack, but the value A-B would be at the top of the stack.

A one-address machine uses an accumulator in order to perform computations. For this problem, you may assume that its ISA allows the following operations:

- LOAD M - Loads the value stored at memory location M into the accumulator.
- STORE M - Stores the value in the accumulator into Memory Location M.
- OP M - Performs the binary operation OP on the value stored at memory location M and the value present in the accumulator. The result is stored into the accumulator (ACCUM = ACCUM OP M).

A two-address machine takes two sources, performs an operation on these sources and stores the result back into one of the sources. For this problem, you may assume that its ISA allows the following operation:

- OP M1, M2 - Performs a binary operation OP on the values stored at memory locations M1 and M2 and stores the result back into memory location M1 (M1 = M1 OP M2).

Note 1: OP can be ADD, SUB or MUL for the purposes of this problem.
Note 2: A, B, C, D, E and X refer to memory locations and can be also used to store temporary results.

a. Write the assembly language code for calculating the expression (do not simplify the expression)

$X = (A + (B \times C)) \times (D - (E + (D \times C)))$

i. In a zero-address machine

PUSH A
PUSH B
PUSH C
MUL
ADD
PUSH D
PUSH E
PUSH D
PUSH C
MUL
ADD
SUB
MUL
POP X

ii. In a one-address machine

LOAD B
MUL C
ADD A
STORE A *[A' = A+(B×C)]*
LOAD D

MUL C
ADD E
STORE E *[E' = E+(D×C)]*
LOAD D
SUB E
MUL A
STORE X *[X = A'×(D-E') = A+(B×C)×(D-E+(D×C))]*

iii. In a two-address machine

MUL B, C *[B' = B×C]*
ADD A, B *[A' = A+B' = A+(B×C)]*
MUL C, D *[C' = D×C]*
ADD E, C *[E' = E+C' = E+(D×C)]*
SUB D, E *[D' = D-(E+(D×C))]*
MUL A, D *[A' = A'×D' = (A+(B×C))×(D-(E+(D×C)))]*
SUB X, X *[X' = X-X = 0]*
ADD X, A *[X' = X'+A' = (A+(B×C))×(D-(E+(D×C)))]*

iv. In a three-address machine like the LC-3b, but which can do memory to memory operations and also has a MUL instruction.

MUL B, B, C *[B' = B×C]*
ADD X, A, B *[X = A+B' = A+(B×C)]*
MUL C, D, C *[C' = D×C]*
ADD E, E, C *[E' = E+C' = E+(D×C)]*
SUB D, D, E *[D' = D-E' = D-(E+(D×C))]*
MUL X, X, D *[X = A'×D' = (A+(B×C))×(D-(E+(D×C)))]*

b. Give an advantage and a disadvantage of a one-address machine versus a zero-address machine.
Advantage: fewer instructions for the same task.
Disadvantage: instructions are longer and more complex, so harder to decode.

Question 6

Variable length instruction v/s fixed length instruction:

Variable instruction length ISAs have more complex decode logic. Variable instruction length ISA programs can be encoded more densely. Variable instruction length ISAs

also generally imply a richer instruction set than that of a fixed length ISA. Since a richer instruction set has more instructions that directly correspond to higher level language programming constructs, the compilation process can be easier.

Question 7

At location x3E00, we would like to put an instruction that does nothing. Many ISAs actually have an opcode devoted to doing nothing. It is usually called NOP, for NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do **nothing** (no change to the state of the machine)! Which of the following three instructions could be used for NOP and have the program still work correctly?

1. 0001 001 001 1 00000
2. 0000 111 000000000
3. 0000 000 000000000

For each of the three that can not be used for NOP, explain why.
The third instruction (never branch) can be used as a NOP. The first instruction (ADD) sets the condition codes based on the value in R1, therefore it is not a NOP.
Additionally, the 2nd instruction (BR always) calculates the result of (PC+2) + Offset during the execution phase. However, it is considered a NOP since the state of the machine is unchanged.

Question 8
Consider the following two LC-3b assembly language programs.

```
    .ORIG x4000          .ORIG x5000
MAIN1 LEA   R3,L1     MAIN2 LEA   R3,L2
A1   JSRR  R3        A2   JMP   R3
    HALT              HALT
    ;                 ;
L1   ADD   R2,R1,R0   L2   ADD   R2,R1,R0
    RET               RET
```

Is there a difference in the result of executing these two programs? If so, what/why is there a difference? Could a change be made (other than to the instructions at Labels A1/A2) to either of these programs to ensure the result is the same?

Yes, there is a difference. The program at x5000 does not save the return address from the subroutine at L2 because it uses a JMP instruction. Thus, that program will not work correctly.

A possible change that could be made:

```
    .ORIG x5000
    LEA   R7, B
MAIN2 LEA   R3, L2
A2    JMP   R3
B     HALT
    ;
L2    ADD   R2, R1, R0
    RET
```

Question 9

Say we have 32 megabytes of storage (1MB = 2^{20} or 1,048,576 bytes), calculate the number of bits required to address a location if

1. the ISA is bit-addressable

   32 MB = $2^5 * 2^{20} * 2^3 = 2^{28}$ bits thus to address each bit you need a 33 bit address

2. the ISA is byte-addressable

   32 MB = $2^5 * 2^{20} = 2^{25}$ bytes thus to address each byte you need a 25 bit address

3. the ISA is 128-bit addressable

   32 MB / 128bits = $2^5 * 2^{20} * 2^3 / 2^7 = 2^{21}$ thus to address every 16 bytes you need a 26 bit address

Question 10

What is the address space of the LC-3b? Of the LC-3?

The address space for both the LC-3b and LC-3 is 2^{16}. The difference is that the LC-3b is byte addressable while the LC-3 is 2 byte addressable (16 bits).