CS 250 Spring 2017 Homework 06 SOLUTION
Due 11:58pm Wednesday, March 01, 2017
Submit your typewritten file in PDF format to Blackboard.

1. What minimum modification to Figure 6.4 to re-design the computer of Chapter 6 as an architecture using 64-bit instructions? Given that instruction size and address size often match, what additional change should be made to Figure 6.4?
   Answer: The circuit in Figure 6.4 implements Fetch_at_the_Default_Next_Instruction. To handle 64-bit instructions the constant increment to the fetch address must be increased from 4 to 8. If the address size is increased from 32 bits to 64 bits, then the progam counter register and the adder in the circuit need to be increased in size to 64 bits.

2. Design an instruction for the processor of Figure 6.9 called *Branch Indirect,* with the mnemonic BRI. BRI sets the Next_Instruction_Pointer equal to the contents of data memory at the sum of an immediate value and a register.
   a. In the style of Figure 6.2, show all information necessary to define BRI.
      Answer: Operation reg_A reg_B dst_reg Offset
      00101 used unused unused used
   b. Execution of a BRI instruction will create what values for the register unit control signals?
      Answer: Write_Enable = False
   c. What operation will be performed by the ALU for BRI?
      Answer: Addition of reg_A with sign-extended Offset
   d. What control signals will be created by BRI for data memory?
      Answer: Read_Enable = true
   e. Which inputs (upper or lower) will be selected by BRI for multiplexers M1, M2, and M3?
      Answer: M1 selects upper; M2 selects lower; and M3 selects upper.
   f. What other Chapter 6 instruction has this same multiplexer selection pattern?
      Answer: None of ADD, LOAD, STORE, and JUMP.

3. How many control signals are generated when an instruction is decoded in Figure 6.9? Assume that the ISA may contain 32 instructions.
   Answer: Each multiplexer needs one bit of address, or control. The Register Unit needs a 1-bit (yes, no) Write_Enable control signal. The Data Memory needs both a 1-bit Read-Enable and a 1-bit Write_Enable. Finally, the ALU needs all five of the opcode bits if there are 32 opcodes. The total is 5 + 3 + 1 + 2 = 11 control signals.

4. Assume that a single chip implements M2 in Figure 6.9. How many pins does this chip have?
   Answer: There are two 32-bit inputs, one 32-bit output, one address input, and, finally, power and ground. Total pins is 99.

5. Define three new instructions in the format of Figure 6.2 as follows. JSR, jump to subroutine, has an opcode field of 00101 and executes by saving the current

default_next_instruction_address in a stack data structure in memory and branching to default_next_instruction_address + Offset. RET, return from subroutine, has opcode 00110 and has automatic access to the address of the top of stack in memory and causes the contents of the top of stack in memory to become the value of next_instruction_address in the fetch circuitry. AND, bitwise logical AND, has opcode 00111.

a. Write the machine code for each line of assembly in the following program snippet. A "snippet" is a few lines of code that may not make logical sense, likely due to missing context. The symbol … confirms that the assembly program exists within a larger context, but that context is not to be part of your answer. Label each line of machine code with its hexadecimal address. **Use a constant-wide font for your answer and put a space between each field in the format of Figure 6.2 to aid readability, knowing that there are no spaces in machine code.** If a format field can contain any bit string, then fill that field with 1. Instead of using … in your answer to denote the unknown contents of memory between main: and MiddleBytes: state on that line of your answer how many intervening instructions could fit in the memory region where the content has not been specified in the assembly program. Note that all registers are global, and thus visible at all times to all instructions.

```
0x003FFFFC                …
0x00400000   main:        JSR MiddleBytes     ; call subroutine MiddleBytes
0x00400004                STORE r1, 0(r5)     ; Memory[r5+0] <- r1
0x00400008                ADD r2, r3, r4      ; r2 <- r3 + r4
0x0040000C                JSR MiddleBytes     ; call subroutine MiddeBytes
 …                         …
0x00400024   MiddleBytes: AND r1,r2,4080      ; r1 <- r2 AND 0x0FF0
0x00400028                RET                 ; return from MiddleBytes
0x0040002C                …
```

Answer:

```
0x00400000   00101 1111 1111 1111 000000000100000
0x00400004   00011 0101 0001 1111 000000000000000
0x00400008   00001 0011 0100 0010 111111111111111
0x0040000C   00101 1111 1111 1111 000000000010100
 A memory gap of unspecified content that could hold
 5 instructions, one each at addresses 0x00400010,
 0x00400014, 0x00400018, 0x00400018, 0x0040001C,
 and 0x00400020.
0x00400024   00111 0010 1111 0001 000111111110000
0x00400028   00110 1111 1111 1111 111111111111111
```

where the JSR at 0x00400000 branches to 0x00400024 = MiddleBytes = default_next_instruction_address + Offset = (0x00400000 + 4) + Offset, therefore Offset = 0x00400024 – (0x00400000 + 0x4) = 0x00000020 = 0x0020, and where the JSR at 0x0040000C branches also to 0x00400024 = MiddleBytes = (0x0040000C + 0x4) + Offset, therefore Offset = 0x00400024 – (0x0040000C + 0x4) = 0x00400024 – 0x00400010 = 0x00000014 = 0x0014.

b. What are the value(s), if any, of all symbolic addresses in the assembly language of part a. of this question?
Answer: The labels main: and MiddleBytes: are symbolic addresses and have the values 0x00400000 and 0x04040024, respectively. Also, the names r1, r2, r3, r4, and r5 are symbolic and correspond to register unit addresses 0001, 0010, 0011, 0100, and 0101, respectively.

c. What are the immediate value(s), if any, in the assembly language of part a. of this question? State any value as a tuple of the form (mnemonic, 0x…). If the value does not fit exactly into an integer number of hexadecimal digits, assume that any extra bits in the hexadecimal notation are zeros. If there are no immediate values, clearly so state in your answer.
Answer: The immediate values are (JSR, 0x0020), (STORE, 0x0000), (JSR, 0x0014), and (AND, 0x0FF0).

d. Are any of the instructions in your machine code position dependent? If not, why not, and is so, which instructions(s) and why?
Answer: The JSR instructions must cause the program counter to update to the specific address of label MiddleBytes:. If any JSR instruction is moved to a different address than shown in the answer to part a., then the immediate value in the offset field of that JSR instruction must be adjusted so that the branch will still be to the address of MiddleBytes:.

e. Did generation of the machine code in your answer to part a. rely on the assembler being two-pass? Describe the action of the two passes with respect to generating the immediate values, if any listed in your answer to part c. of this question.
Answer: Machine code generation did rely on both passes of the assembler.
There is a forward symbolic reference from the instruction at main:, JSR MiddleBytes, to the subroutine starting at MiddleBytes:. The first pass of the assembler would have eventually generated a filled-in symbol table showing the correspondence of the symbol MiddleBytes: to the memory address 0x00400024. This information would be used during the second assembler pass to compute the necessary offset from the addresses of both JSR instructions to the start of the subroutine at MiddleBytes:.
The generation of the immediate field bit strings for the STORE and AND instructions could have happened during the first pass because all the necessary information to generate these bit fields exists within the assembly instruction itself, but doing so would require maintaining a data structure of partially built machine code for the second assembler pass to complete. A better design choice is for the first pass to only create a complete symbol table with all symbol to memory address correspondences and then in the second pass generate each machine code instruction bit string one at a time, completely.

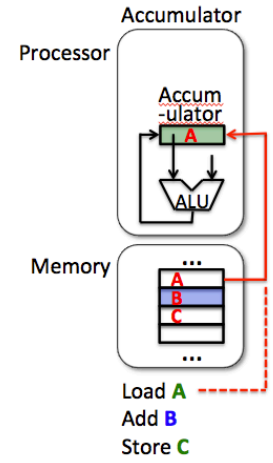
6. Using Lecture 13, Slide 15 as a reference, draw a sequence of three diagrams showing the movement of data through a 1-address machine executing the three lines of assembly corresponding to C=A+B. You answer should contain one diagram for each line of code.

The architectural part of each of the three diagrams should be identical. Then, each diagram also shows the location of each data item mentioned in the corresponding line of code and has arrows showing all data item movement(s) resulting from the execution of that line of code. See Lecture 13 Slides 6 through 9 for an example of the identical question answered for a 0-address architecture.

Answer: Assuming that A, B, and C has all been declared and allocated memory space.

Code for C=A+B for 1-addr. machine

Assumes A, B, C all belong in memory and that A, B not destroyed



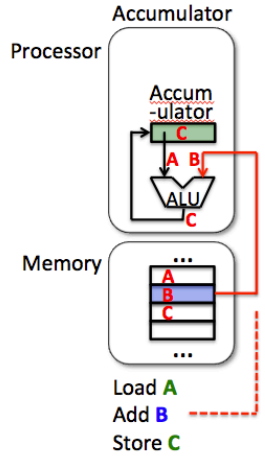
Assembly instruction LOAD copies the contents of a memory location into the Accumulator register.

LOAD A uses the pointer A to point at a memory location, where the contents, also called A, are copied to the accumulator.

An accumulator is a register connected to perform the function of providing one operand to the ALU and receiving the corresponding result from the ALU. When an item is to be loaded into the accumulator, the current contents of the accumulator are overwritten, so programs must take this behavior into account.

Code for C=A+B for 1-addr. machine

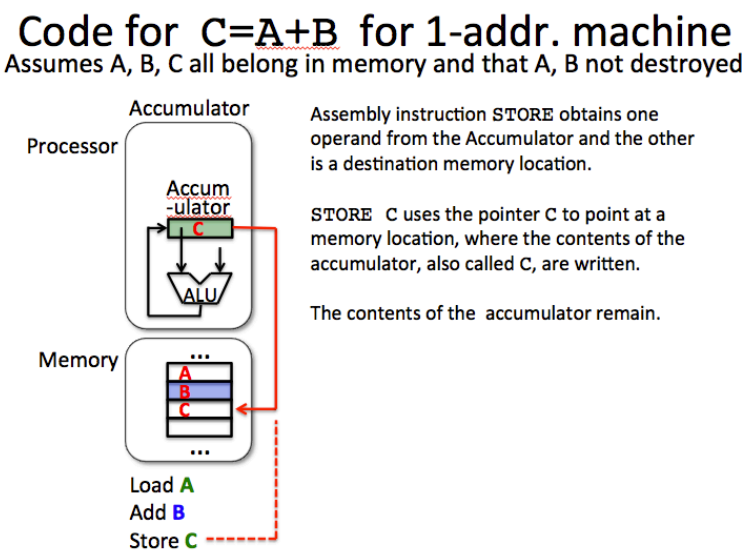
Assumes A, B, C all belong in memory and that A, B not destroyed



Assembly instruction ADD obtains one operand from the Accumulator and the other from a memory location.

ADD B uses the pointer B to point at a memory location, where the contents, also called B, are added to the accumulator producing result C.

Result C then overwrites the value A that is in the accumulator, so programs must take this behavior into account.



7. Which addressing modes of Figure 7.6 are
   a. impossible, and which are possible, for a machine with 32-bit instructions and a 32-bit address? Give a reason for any impossible mode.
   Answer: This machine requires 32-bit memory addresses and, so, modes 3 and 5, which store a memory address within an instruction, are not possible. Modes 1, 2, and 4 are possible.
   b. perhaps possible for a machine with 64-bit instructions and 16 Gbytes of byte-addressed memory? Give reasons.
   Answer: Addressing all of 16 GBytes of byte-addressed memory requires at least a 34-bit address (2^34 = 2^4 x 2^30). If the machine fills any high-order bits of an address with zeros, and if this machine uses 64-bit instructions, then there could be room within an instruction for a 34-bit direct address field as well as the required opcode field and destination or source register field for the load or store represented by addressing modes 3 and 5.