CSC 317 Homework #2 – Due 5/1/2015

1. Introduction

You have been hired by the CEO of Old Fashion Computing to help develop the company's first offering, to be named the B17. The B17 is a simple accumulator-style architecture. Your supervisor has decided that your first task will be to develop a CPU simulator for the B17.

---

2. CPU Architecture

The B17 is a 24-bit word-addressable accumulator architecture. It supports up to 64 instructions, up to four addressing modes, and 2^{12} (4096) words of memory, as specified below.

2.1. Main Memory

Main memory consists of 4,096 words, each of which is 24 bits wide. Addresses are 12 bits wide, and range from `000` through `FFF`. Within the word, bits are numbered from right to left, with bit 0 the least significant bit and bit 23 the most significant bit. The memory interface transfers one word of data at a time. Communication with memory is via two registers, `MAR` and `MDR`, described below.

2.2. Registers, Buses, Other Logic

All arithmetic and logical operations are performed between an accumulator register (named `AC`) and the contents of a memory location.

The following components exist in the B17 architecture:

Item	Size	Name	Contents/Usage
`MAR`	12	Memory Address Register	Address of the memory location which is to be loaded from or stored into.
`IC`	12	Instruction Counter	Address of the next instruction to be fetched, decoded, and executed.
`X`*n*	12	Index Registers	Four registers (X0-X3); contain values to be used in calculating memory addresses.
`ABUS`	12	Address Bus	Used when addresses are to be moved.
`MDR`	24	Memory Data Register	Data to be written into, or data most recently read from, memory.
`AC`	24	Accumulator	The accumulator register.
`ALU`	24	Arithmetic-Logic Unit	Performs computations.
`IR`	24	Instruction Register	Instruction being decoded and executed.
`DBUS`	24	Data Bus	Used when data and instructions are to be moved.

You may add other non-visible registers to the CPU if they are required (e.g., to contain constant values which will be used with the ALU). However, your architecture must be reasonable - you should strive for the smallest number of additional modules possible in your design, and added modules should provide functionality that cannot be implemented with the standard register set.

2.3. The Instruction Set

2.3.1. Instruction format

The B17 uses fixed-length instructions. Instructions have the following format:

`ADDR` is a twelve-bit operand address (bits 23-12).
`OP` is a six-bit opcode (bits 11-6).
`AM` is a six-bit addressing mode (bits 5-0).

2.3.2. Addressing Modes

The B17 addressing mode (AM) field consists of a four-bit mode specifier and a two-bit index register number. Mode bits are interpreted as follows:

Bits 5-2	Name	Interpretation
0000	Direct	EA is the contents of the address field.
0001	Immediate	There is no EA - the operand data is the sign-extended address field contents.
0010	Indexed	EA is the sum of the address field and the specified index register.
0100	Indirect	Address field contains the address of an indirect word in memory; the EA is the upper 12 bits of that word.
0110	Indexed Indirect	Perform indexing; the result is the address of an indirect word, whose upper 12 bits is the EA.
other	Illegal	Causes an ``illegal addressing mode'' error

2.3.3. Instructions

Each six-bit opcode is interpreted as a two-bit category indicator (miscellaneous, memory, ALU, or transfer) and a four-bit operation specifier; thus, there can be up to 16 instructions in each category.

The following instructions are defined; entries marked with ? are reserved for future use. The indicated bit positions are within the instruction word.

	Bits 11-10			
Bits 9-6	**00**	**01**	**10**	**11**
0000	HALT	LD	ADD	J
0001	NOP	ST	SUB	JZ
0010	?	EM	CLR	JN
0011	?	?	COM	JP
0100	?	?	AND	?
0101	?	?	OR	?
0110	?	?	XOR	?
0111	?	?	?	?
1000	?	LDX	ADDX	?
1001	?	STX	SUBX	?
1010	?	EMX	CLRX	?
1011	?	?	?	?
1100	?	?	?	?
1101	?	?	?	?
1110	?	?	?	?
1111	?	?	?	?

Instruction mnemonics have the following meanings:

Mnem.	Description	Legal Addressing Modes
HALT	Halt the machine.	Ignored.
NOP	Do nothing.	Ignored
LD	Load the accumulator from memory.	All.
ST	Store the accumulator into memory.	All except Immediate.
EM	Exchange the accumulator with memory.	All except Immediate.
LDX	Load the specified index register from the upper half of a memory word.	Direct, Immediate.
STX	Store the specified index register into the upper half of a memory word.	Direct.
EMX	Exchange the specified index register with the upper half of a memory word.	Direct.
ADD	Add memory to the accumulator.	All.
SUB	Subtract memory from the accumulator.	All.
CLR	Clear the accumulator.	Ignored.
COM	Complement the accumulator.	Ignored.
AND	AND memory to the accumulator.	All.
OR	OR memory to the accumulator.	All.
XOR	XOR memory to the accumulator.	All.
ADDX	Add memory to the specified index register.	Direct, Immediate.
SUBX	Subtract memory from the specified index register.	Direct, Immediate.
CLRX	Clear the specified index register.	Ignored.
J	Jump to the specified memory address.	All except Immediate.
JZ	Jump to the memory address if the accumulator contains zero.	All except Immediate.
JN	Jump to the memory address if the accumulator contains a negative number.	All except Immediate.
JP	Jump to the memory address if the accumulator contains a positive number.	All except Immediate.

J, JZ, JN, and JP instructions cannot be used with Immediate mode.

HALT and NOP instructions ignore the addressing mode field entirely, and do not generate invalid mode errors, regardless of the contents of that field.

3. Simulator Requirements

3.1. Instruction Subset

You will implement the machine as described above, except for the following items:

- You will not implement the index register instructions (LDX, etc.)
- You will not implement indexed and indirect addressing.

3.2. Execution Record

For every instruction your simulator executes, print a trace line. Each trace line should contain the following:

- the address of the instruction (three hex digits)
- the instruction itself (six hex digits)
- the instruction mnemonic (four characters)
- the EA being used by this instruction if it uses a memory address (three hex digits), or `IMM` if it specifies an immediate operand, or three spaces
- the contents of AC and the four index registers *after* the execution of the instruction

The instruction address should be followed by a colon (:); there should be two spaces between each pair of fields. This cycle continues until the machine halts, at which time your simulator should print an appropriate halt message (see below).

Here is a sample object file:

```
50 1 000000
c4 5 050404 200800 300800 102840 050c00
101 2 300 9
200 1 30
300 1 10
c4
```

Here is the output from your simulator run against that object file:

```
0c4:  050404  LD    IMM  AC[000050]  X0[000]  X1[000]  X2[000]  X3[000]
0c5:  200800  ADD   200  AC[000080]  X0[000]  X1[000]  X2[000]  X3[000]
0c6:  300800  ADD   300  AC[000090]  X0[000]  X1[000]  X2[000]  X3[000]
0c7:  102840  SUB   102  AC[000087]  X0[000]  X1[000]  X2[000]  X3[000]
0c8:  050c00  J     050  AC[000087]  X0[000]  X1[000]  X2[000]  X3[000]
050:  000000  HALT       AC[000087]  X0[000]  X1[000]  X2[000]  X3[000]
Machine Halted - HALT instruction executed
```

Follow this output format *precisely*; sample output files (see below) will help you determine the correct format.

3.3. Object File

The object code file name is supplied as a command-line argument to your program.

3.4. Halting

When your program halts, print `"Machine Halted - "`, followed by one of the following messages, as appropriate:

Message	Reason	Action
`HALT instruction executed`	A HALT instruction was executed.	Stop the simulation.
`undefined opcode`	An opcode which is not defined (see above) was encountered.	Print `????` for the mnemonic in the trace; perform as a `NOP`; stop the simulation.
`unimplemented opcode`	An opcode which is not implemented (see above) was encountered.	Print the mnemonic in the trace; perform as a `NOP`; stop the simulation.
`illegal addressing mode`	An invalid mode was specified, or a mode which is illegal for this instruction.	Print `???` for the memory address in the trace; perform as a `NOP`; stop the simulation.
`unimplemented addressing mode`	An addressing mode which is not implemented (see above) was encountered.	Print `???` for the memory address in the trace; perform as a `NOP`; stop the simulation.

Check for errors in the order shown above - e.g., if the opcode is undefined or unimplemented, don't check the addressing mode. Note that the full instruction trace line is printed for each instruction fetched, regardless of any error that may occur, and the error message is printed *after* the trace line.

---

4. Turning In Your Solution

4.1. General Information

The program must be submitted by the end of the day on the specified due date (i.e., no later than 11:59:59pm that day). The files need to be tarred and gzipped prior to submission. Please empty the directory prior to tar/zip. [You will tar up the directory containing the files: `tar -czf prog2.tgz prog2 .`]

Submission contents:

- Documentation
    - Complete description of the implementation.
        - You must document the author of each code segment.
        - You must document the percentages of effort and these must add up to 100%
    - Complete description of the testing.
    - Complete description of what the system requirements are, how to build, how to run.
- Main program.
- Any required external functions `*.c, *.h`.
- Any include files you use (other than the standard ones).
- You will need to produce a Makefile to build the executable:

The file which contains your `main()` function *must* be named `b17.C`. Your program will be invoked with the command line

```
b17  prog.obj
```

where `prog.obj` is the name of the file containing the object program to be simulated.

5.2. Submitting Your Solution

One member of your team will use the submit page to submit your program. The other members will email me the percentage efforts of the team.
