18-240: Fundamentals of Computer Engineering

Carnegie Mellon

Using the P18240 Assembler and Simulator

updated: 9 April 2014

This tutorial will show you how to use several tools to work with your P18240 assembly language programs. The assembler, **as240**, will convert your assembly code into machine language. The simulator, **sim240**, will allow you to execute your machine language program as if it was running on the P18240 CPU. Using it, you can single step (either instruction-at-a-time or state-at-a-time) through your program and observe the changes to the register file, special registers, datapath registers and memory.

Both programs run on the linux servers and are hosted in the class AFS space. They are both perl programs, so you could copy it to your own computer and run it there if you have perl installed. Both programs are found in **/afs/ece/class/ece240/bin**.

The assembler is a fairly easy tool to work with. You feed it an assembly language file(which must have a .asm extension) and it spits out machine code. Like so:

```
>as240 fibo.asm
```

The assembler produces two different files, fibo.list and memory.hex. The later is merely a human-readable list of the values in the program's memory image[1]. The .list file contains the memory image, but it also includes the corresponding address values and the assembly instructions.

The simulator is an interactive tool, so has more power but takes a bit more effort on your part. Start the simulator by specifying the list file:

```
>sim240 fibo.list
```

You will be rewarded with the simulator prompt:

```
>
```

The simulator has loaded your file and is ready to do stuff at your command. Why don't you try it out for yourself? You can find the **fibo.asm** file at **/afs/ece/class/ece240/doc/P18240/Tutorial**.

```
        .ORG    $0000   ; execution starts at 0
        LDI     R0, $0  ; initialize the stack
        LDSP    R0
        BRA     start   ; jumps to main routine

        .ORG    $100
start   LDI     R0, $0  ; R0 <- $0 (immediate)
        LDI     R1, $1  ; R1 <- $1 (immediate)
```

[1] The memory.hex file will be useful when you synthesize the P18240 in Lab4.

```
loop    CMI     R1, $D  ; R1 - D
        BRZ     done    ; branch if R1 == D (R1 == 13)
        ADD     R0, R1  ; R0 <- R0 + R1
        ADD     R1, R0  ; R1 <- R1 + R0
        BRA     loop    ; branch always ("go to")

done    STOP            ; all done
```

The simulator will respond to a fair number of commands. Take a look through the list by using the help command.

> help

The most important command is probably the **quit** command (you can also use **exit**).

Try out the simulator commands. Start with **run**, which will execute the entire program, until it finds a **STOP** instruction. As each instruction is executed, it will print the state of the P18240, thus providing you with a trace of the program execution.

Use the **reset** command to clean up and try again. In effect, **reset** will reload the simulation file and change the registers back to their starting values.

You can take a look at the changes brought about by each instruction, one-by-one, with the **step** command. After executing a single instruction, **step** will print out a one-line state summary so you can find out what changed. Go ahead and step through a few instructions and watch the changes to **PC**, **IR**, **R0** and **R1**.

The **ustep** command does a similar process, except that it only executes a single state transition in the FSM. You will have to ask to see the state (with **state?**) or registers (with ***?**) after each **ustep** command[2]. Watch the state field in the summary and you will see (with successive micro command invocations) the state go from FETCH → FETCH1 → FETCH2 → DECODE, etc. This is especially useful to see changes in the MAR and MDR.

You can modify the simulated operation of the P18240 by changing the registers of the datapath or changing the values in memory. Use the **IR=** command to put a different value in the IR (for instance). Use **M[addr]=** to change memory contents:

> IR=0c08

> M[0102]=1492

Experiment with these and other simulation commands. Then do the following:

1. Assemble **fibo.asm** by hand to produce the binary memory listing. Yes, I said "by hand." Make sure to show your work, which includes several pieces of information about each line: the addressing mode, instruction format, and binary encoding of the first word. For long instruction formats, the second word can be in hex. Also, include your symbol table (list of what values each label maps to).

---

[2] An enhancement request has been filed.

2. Assemble **`fibo.asm`** using the assembler (**`as240 fibo.asm`**) to check your hand assembled memory listing. The machine language is either in **`memory.hex`** or **`fibo.list`**. (in different formats, **`fibo.list`** is probably easier to manage).

3. Simulate the execution of **`fibo.asm`**. In the table below, provide the values of the PC, MDR, MAR, IR, CCR (condition code register or ZCNV codes), and registers R0 and R1 from the general purpose register file **after** each instruction is executed the first time only (including the stop). Give each of the values in 4-digit hex representation (for 16 bits), except for the condition codes (just give those in binary).

	PC	IR	ZNCV	MAR	MDR	R0	R1
`BRA START`							
`LDI R0, $0`							
`LDI R1, $1`							
`CMI R1, $D`							
`BRZ DONE`							
`ADD R0, R1`							
`ADD R1, R0`							
`BRA LOOP`							
`STOP`							

4. Write a few P18240 Assembly programs of your own to try out different things.