Prof. Mark D Shattuck
Physics 39907 Computational Physics
September 5, 2024

Problem Set 2

Question 1. *4-bit Assembly:* Consider a theoretical 4-bit CPU shown in 1 with the instruction set shown in Table 1. This figure is taken from the Crash Course Computer Science series episodes 7 and 8. These videos are good resources for this question (although there are a few small differences in the implementation between Table 1 and the instructions in the videos. CPU4sim.m is a MATLAB tool for simulating the CPU. CPU4sim allows you to program the CPU using a simple assembly programming language described in Table 1. In this language the code:

```
LDA 14
LDB 15
ADD A B
STA 13
HLT
```

tells the CPU to add the value stored in memory location 14 to the one in location 15 and store the result in memory location 13. To input this program into CPU4sim you define a variable called `prog`. Here is the pre-programmed example from the first few lines of CPU4sim.m:

```
function CPU4sim(prog)
%% Clear figure
clf;

%% Program
if(¬exist('prog','var') || isempty(prog))

  [prog{1:16}]=deal('');  % set all 16 locations to empty

  % code
  prog{1}='LDA 14';
  prog{2}='LDB 15';
  prog{3}='ADD A B';
  prog{4}='STA 13';
  prog{5}='HLT';

  % data
  prog{15}=45;
  prog{16}=13;

end
...
```

The variable `prog` is a cell array so it can store different types of data. If the type of data is a string of characters like the first element `prog{1}='LDA 14';` then the string is converted to an 8-bit instruction using Table 1. If the entry is a number like `prog{16}=13;` then the number is stored directly into memory. The index `n` in `prog{n}` determines which memory location. So `prog{7}='SUB C D';` would store the instruction for `SUB C D` in the 7th memory location. The CPU has 16 memory locations addressed from

0–15 so the 7th location is $ADDR = 6$. To program the simulator update the `%% Program` section with new instructions and run the code. When the simulator starts all of the registers are random so a RESET is required before begining the FETCH/DECODE/EXECUTE cycle. In class we worked on writing a program to find the remainder when the number in $ADDR = 14$ is divided by the number in $ADDR = 15$. The result is stored in $ADDR = 13$. We had the idea to add at the end to get the final answer. This only works for signed integers, which we do not have here. Try to think about how to correct this. (Spolier next!) One solution is to store the intermediate answers. Here is a corrected program:

```
00:LDA 14
01:LDB 15
02:SUB A B
03:JPN 06
04:STA 13
05:JMP 02
06:HLT;
```

The line numbers are there to help with the jump commands but are not allowed when programming CPU4sim. Also the spaces are important in CPU4sim. So `SUB A B` is not the same as `SUBAB`. To program CPU4sim to calculate the remainder use the following:

```
%% Program
if(¬exist('prog','var') | | isempty(prog))
    [prog{1:16}]=deal('');

    % code
    prog{1}='LDA 14';
    prog{2}='LDB 15';
    prog{3}='SUB A B';
    prog{4}='JPN 06'; %JPN 7th
    prog{5}='STA 13';
    prog{6}='JMP 02'; %JMP 3rd
    prog{7}='HLT';

    % data
    prog{15}=45;
    prog{16}=13;

end
```

The comments like `%JPN 7th` are there to make it clear to ”jump if negative” to the 7th memory location which is $ADDR = 06$.

(1) Write a program in this assembly language to find the maximum of two numbers stored in $ADDR = 15$ and $ADDR = 14$ and store the result in $ADDR = 13$. You can test it in CPU4sim. If CPU4sim gets stuck in a loop, and the reset button will not stop it, then you can press ctrl-c in the command window to stop it. If the font is too big or small you can use this command `set(findall(gcf,'-property','FontSize'),'FontSize',10)` to change it. 8 is the default size. If you want to change the data in RAM while CPU4sim is running you can delete everything in the location and type a single number and press return.

(2) Write a program to divide the number in $ADDR = 14$ by the number in $ADDR = 15$ and store the result in $ADDR = 12$ and the remainder in $ADDR = 13$.

(3) Write a program to multiply two numbers who product is less than or equal to 255. So $17 \times 12 = 204$ is okay, but $16 \times 16 = 256$ is not. Be sure to think about edge cases like $0 \times 0 = 0$.

4-bit opcode	Name	Address or Registers	Example	Description
0000 (0)	NOP	*none*	N O P	No operation but (increment IP)
0001 (1)	LDA	*4-bit ADDR*	L D A 1 2	Load contents of *ADDR* into register A
0010 (2)	LDB	*4-bit ADDR*	L D B 1 4	Load contents of *ADDR* into register B
0011 (3)	LDC	*4-bit ADDR*	L D C 0 3	Load contents of *ADDR* into register C
0100 (4)	LDD	*4-bit ADDR*	L D D 0 0	Load contents of *ADDR* into register D
0101 (5)	STA	*4-bit ADDR*	S T A 1 2	Store contents of register A into *ADDR*
0110 (6)	STB	*4-bit ADDR*	S T B 0 7	Store contents of register B into *ADDR*
0111 (7)	STC	*4-bit ADDR*	S T C 0 1	Store contents of register C into *ADDR*
1000 (8)	STD	*4-bit ADDR*	S T D 1 0	Store contents of register D into *ADDR*
1001 (9)	JMP	*4-bit ADDR*	J M P 1 5	Jump to 4-bit memory address *ADDR*
1010 (10)	JPO	*4-bit ADDR*	J P O 1 0	Jump to *ADDR* if O-flag (Overflow) is set
1011 (11)	JPN	*4-bit ADDR*	J P N 1 2	Jump to *ADDR* if N-flag (Negative) is set
1100 (12)	JPZ	*4-bit ADDR*	J P Z 0 5	Jump to *ADDR* if Z-flag (Zero) is set
1101 (13)	ADD	*R1 R2*	A D D A C	R1=R1+R2 e.g., store A+C in A (Flags:OZ)
1110 (14)	SUB	*R1 R2*	S U B C B	R1=R1-R2 e.g., store B-C in B (Flags:NZ)
1111 (15)	HLT	none	H L T	Halt operation and (decrement IP)

TABLE 1. Instruction set for a theoretical 4-bit CPU. *ADDR* represents one of the 16 4-bit memory addresses. For example, the full 8-bit instruction to store register A in memory location 12 (`LDA 12`) is `0b01101100`. *Rn* where n is 1 or 2 represent 2-bit addresses of registers A=`0b00`, B=`0b01`, C=`0b10`, D=`0b11`. For example the full 8-bit code for (`ADD D A`) is `0b11011100`.

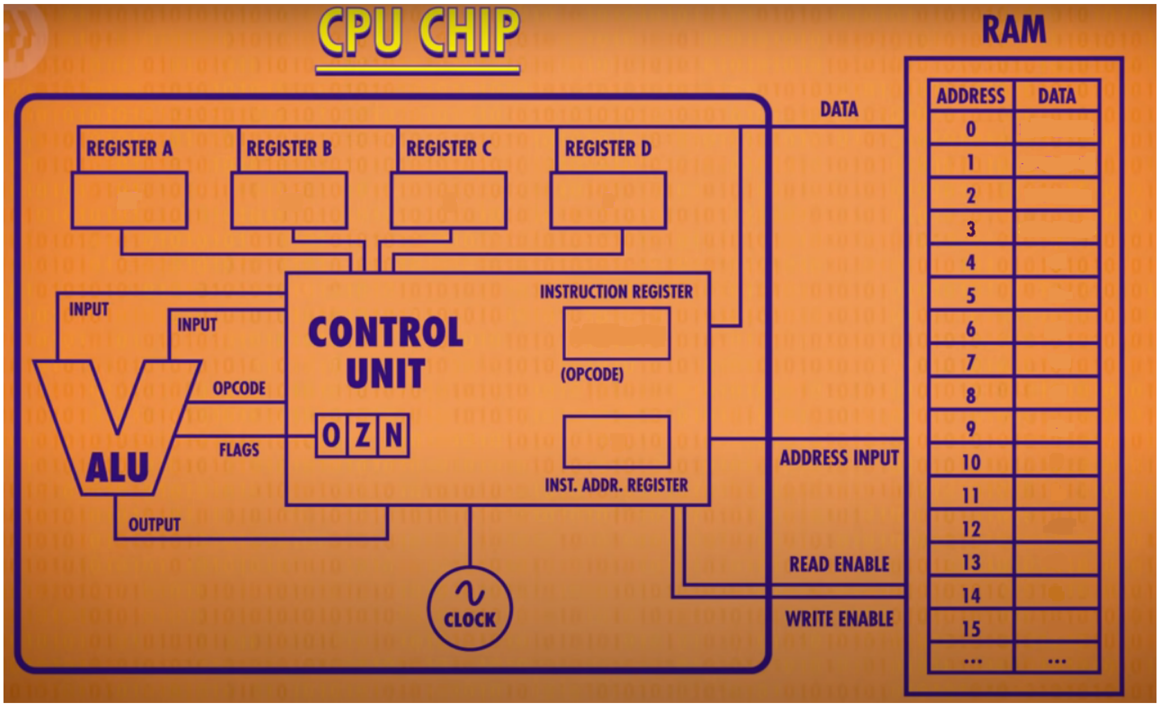

Figure 1. Layout for a theoretical 4-bit CPU.

Question 2. Introduction to MATLAB: Go to Introduction to MATLAB and download the MATLAB Primer. Go through the steps and write up 3 out of the 7 problems at the end. If you are already a MATLAB expert you could do parts or all of the Advanced (Particle Tracking) exercises here.