CS 221B Computer Architecture

Week 4: Introduction to PEP/6

Fall 2000

Course Schedule

W1	Sep 11- Sep 14	Introduction	
W2	Sep 18- Sep 21	Information Representation (1)	(Chapter 3)
W3	Sep 25- Sep 28	Information Representation (2)	(Chapter 3)
W4	Oct 2- Oct 5	Computer Architecture Pep/6	(Chapter 4)
W5	Oct 9- Oct 12	Assembly Language	(Chapter 5)
W6*	**Oct 16- Oct 19**	**High-Level Languages (1)**	**(Chapter 6)**
W7*	**Oct 23- Oct 26**	**High-Level Languages (2)**	**(Chapter 6)**
	Midterm*		
W8	Oct 30- Nov 2	Instruction Sets	(Tanenbaum)
W9	Nov 6 - Nov 9	Devices & Interrupts	(Chapter 8)
W10*	**Nov 13- Nov 16**	**Storage Management**	**(Chapter 9)**
W11	Nov 20- Nov 23	Combinational Logic	(Chapter 10)
W12	Nov 27- Nov 30	Sequential Logic & Review	(Chapter 11)

**Marked classes will be taught by Dave Dove, same place same slot*

Week 4: Plan

- **Introduction to PEP/6**
- **Instruction Set**
- **Programming PEP/6 in machine language**
- **PEP/6 Operating System**

- **Readings**
  - Chapter 4
  - Next week: Chapter 5 of book

Announcements

- **Reading for next week is book c5**
- **Tutorial tomorrow 13:30 on PEP/6 basics.**
  - see web home page.
- **Half of you will come here for Q&A session about assignment 1 with TAs**
- **Other half will proceed for tutorial to GH 230 235**
  - Fill up 230, then proceed filling up 235
  - TAs will be present there as well
- **After 20 minutes, groups swap.**
- **Let's split up the class now.**
  - Family name >J proceed to Goodwin first
  - Family name < J come here first

I.
PEP/6 Execution Cycle and First Program

What is PEP/6

- **PEP/6 is a virtual computer running on your real computer as a software program**

- **It was designed to teach you the essence of programming machine code and assembly**
  - Without becoming too hardware dependent.
  - Without having to crash a unix box.

- **We will use the PEP/6 simulator throughout the course as an instructional tool**

- **It's up to you to relate this to real processors.**

Opcodes

- **These specify which operation should be done**
  - An ADD, an AND or whatever
- **The opcode is decoded by the CPU into a sequence of logic gate operations**
- **In PEP/6, the opcode is 5 bits, total of 32 opcodes**
  - Some are left for future
  - Table 4.1 page 126 in book shows all opcodes
- **We will go through some of them**

NOT

00111 *One's Complement of Register R*

- Inverts bits in index register or accumulator.
- Affects N Z status bits

Example

00111 0 01
Not A ignored

Before A: FFF0 (h) *After* A: 000F (h)

NZVC: 0100 result is zero

Byte Instructions (1)

01010 *Load Byte Operand into Register R*

- Byte instruction: operates on single byte at a time.
- Loads byte from memory into right half of index register or accumulator, leaves left half alone.
- N and Z bits are affected.

Example:

- 01010 0 01 0000 0000 0100 1010
  Load Byte A direct addressing operand address 004A (h) in main memory

Before A: 036D (h) *After* A: 03E2 (h)
Mem[004A]: E2 (h) Mem[004A]: E2 (h)

NZVC: 0000

Byte Instructions (2)

01011 *Store Byte Operand from Register R to Operand*

- Byte instruction: operates on single byte at a time.
- Stores byte from right half of index register or accumulator into memory, ignores left half.
- N and Z bits are affected.

Example:

- 01011 0 01 0000 0000 0100 1010
  Load Byte A direct addressing operand address 004A (h) in main memory

Before A: 92BC (h) *After* A: 92BC (h)
Mem[004A]: F0 (h) Mem[004A]: BC (h)

NZVC: 0000

Byte Instructions: I/O (3)

11011 *Character input to Operand*

- Byte instruction: operates on single byte at a time.
- Takes next ASCII character from the input device and stores binary code in a byte in memory.
- Ignores register specifier.

Example: Read 'W'

- 11011 0 01 0000 0000 0100 1010
  Input Char - direct addressing operand address 004A (h) in main memory

Before Input 'W' (h) *After* Input ---
Mem[004A]: F0 (h) Mem[004A]: 57 (h)

Byte Instructions: I/O (4)

11100 *Character output from Operand*

- Byte instruction: operates on single byte at a time.
- Sends content of byte in memory to output device, which prints corresponding character.
- Ignores register specifier.

Example: Read 'W'

- 11100 0 01 0000 0000 0100 1010
  Ouput Char - direct addressing operand address 004A (h) in main memory

Before Ouput --- (h) *After* Output 'W'
Mem[004A]: 57 (h) Mem[004A]: 57 (h)

Pep/6 Von Neumann Machine

- **Pep/6 is a Von Neumann Machine**
  - storing not only data, but also a set of instruction inside the computer memory
- **Von Neumann Execution Cycle**
  - In order to fetch instructions from memory for execution a certain procedure is iterated:
  - This is called the Von Neumann Execution Cycle

```
Load the machine language program
Initialize PC and SP
repeat
       Fetch the next instruction
       Increment PC
       Execute the instruction fetched
until the stop instruction is executed.
```


The Execution Cycle

This cycle consists of 4 operations : fetch, increment, execute and repeat.

1. Load the machine program into the main memory.

The first instruction is placed in address 0000 (hex). All the following instructions are placed consecutively. The actual location of a given instruction depends on all previous instructions.

2. To initialize the program counter (PC) the value of the memory address of the first instruction in the memory is assigned to PC.

In our situation (Pep/6) this value is 0000. The stack pointer (SP) is initialized to the value at the memory address 0FF8.

3. The first operation in the cycle is fetch.

This is done by CPU looking at the value in PC and interpreting it as a memory location address. One byte from that address in the main memory is copied into the first byte of the instruction register (IR). The CPU examines that opcode to determine the actual length of that instruction. If it is not unary then the operand is fetched and stored into last two bytes of IR.

The Execution Cycle

4. The second operation is increment.

The CPU adds 0001 to the PC if the instruction just fetched was unary (i.e. 1 byte long) and adds 003 if the instruction was not unary.

5. The third operation is execute.

The instruction as it appears in IR is examined for it's opcode and executed.

6. The fourth operation is repeat.

The CPU returns to the fetch operation unless the last operation was a stop instruction, or some illegal operation was attempted.

Execution Cycle in PEP/6

- **So in PEP/6, here is what happens:**

```
Load the machine language program into memory
starting at address 0000
PC := 0000
SP := mem [0FF8]
repeat
      Fetch the next instruction specified by
PC
      if the instruction is unary then
            PC := PC + 1
      else
            PC := PC + 3
      Execute the instruction fetched
until the stop instruction is executed or an
illegal operation is attempted.
```


Our First Program

- **The first program will display three characters 'fun' on the output device.**
  - We assume that the ASCII codes for those three characters is our data
  - This data immediatelly follows our list of instructions.
  - The ASCII code for characters 'fun' in hex is 66, 75 and 6E.

The Code

- **Depending on the instruction (unary or binary), we will use 1 or 3 columns to divide memory.**
- **The leftmost column contain the starting address of the first byte of each instruction.**
  - Remember that the instructions can be 1 or 3 bytes in length.
- **The right-hand side contains the contents of the main memory.**
  - The loader of Pep/6 always loads application programs at the top of the main memory at address 0000 hex.

0000	1110	0001	0000	0000	0000	1010
0003	1110	0001	0000	0000	0000	1011
0006	1110	0001	0000	0000	0000	1100
0009	0000	0000				
000A	0110	0110				
000B	0111	0101				
000C	0110	1110				

The Code In Hex

address *content*

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00	00	
000A	66		
000B	75		
000C	6E		

Tracing the State of PEP/6 (1)

1) *This is the initial state of the relevant parts of the Pep/6 before the program executes. Each '?' stands for unknown hex value.*

CPU

PC: ????

IR: ??????

Main Memory

0000 ?? ?? ??
0003 ?? ?? ??
0006 ?? ?? ??
0009 ??
000A ??
000B ??
000C ??

Output device nothing is shown

Bus no activity

Tracing the State of PEP/6 (2)

2) *Loading of the application program into the main memory*

CPU

PC: ????

IR: ??????

Main Memory

0000 E1 00 0A
0003 E1 00 0B
0006 E1 00 0C
0009 00
000A 66
000B 75
000C 6E

Output device nothing is shown

Bus no activity

Tracing the State of PEP/6 (3)

3) *Program counter (PC) is initialized to the address of the starting byte of the program - in Pep/6 it is always 0000*

CPU

PC: 0000

IR: ??????

Main Memory

0000 E1 00 0A
0003 E1 00 0B
0006 E1 00 0C
0009 00
000A 66
000B 75
000C 6E

Output device nothing is shown

Bus no activity

Tracing the State of PEP/6 (4)

4) *Fetch first instruction: load initial byte of instruction into instruction register (IR). CPU determines if unary instruction (fetch is complete) or binary instruction (fetch the remaining 2 bytes into the IR).*

CPU

PC 0000

IR E1000A

Main Memory

0000 E1 00 0A
0003 E1 00 0B
0006 E1 00 0C
0009 00
000A 66
000B 75
000C 6E

Output device Nothing is shown

Bus Contents of first instruction sent to register IR.

Tracing the State of PEP/6 (5)

5) *Program counter (PC) is incremented to show the starting byte address of the next instruction: if the instruction just fetched was unary, then PC is incremented by 0001, otherwise it is incremented by 0003.*

CPU

PC: 0003

IR: E1000A

Main Memory

0000 E1 00 0A
0003 E1 00 0B
0006 E1 00 0C
0009 00
000A 66
000B 75
000C 6E

Output device nothing is shown

Bus no activity

Tracing the State of PEP/6 (6)

6) *The instruction in the IR register is executed: this results in appearance of the first character on the output device.*

CPU

PC: 0003

IR: E1000A *11100 (E1h): character output from direct address A*

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device Character 'f' appears on the output device

Bus: ASCII code for the character 'f' is sent from memory location 000A to output device.

Tracing the State of PEP/6 (7)

7) *Another fetch. As before based on the address in the PC 1 byte is fetched into the IR and the length of the instruction is determined. In our case the remaining 2 bytes are also copied to the IR.*

CPU

PC: 0003

IR: E1000B

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device Character 'f' is visible

Bus Contents of 3 bytes from main memory (starting location 0003) are sent to IR in CPU.

Tracing the State of PEP/6 (8)

8) *Program counter (PC) is incremented to show the starting address of the next instruction - in our case by 003.*

CPU

PC: 0006

IR: E1000B

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device Character 'f' is visible

Bus No activity

Tracing the State of PEP/6 (9)

9) *Execute the instruction in IR. Sending of ASCII character code from memory location 000B to the output device.*

CPU

PC: 0006

IR: E1000B *11100 (E1h): character output from direct address B*

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device New character 'u' appears after character 'f'

Bus ASCII code from memory location 000B sent to output device

Tracing the State of PEP/6 (10)

10) *Fetch next instruction.*

CPU

PC: 0006

IR: E1000C

Main Memory:

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device Characters 'fu' are still visible from the previous instructions execution.

Bus instruction from address 0006 is sent to the IR

Tracing the State of PEP/6 (11)

11) *Increment Program Counter.*

CPU

PC: 0009

IR: E1000C

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device Characters 'fu' are still visible

Bus No activity

Tracing the State of PEP/6 (12)

12) *Execution of the instruction in IR - 1 byte of data is sent from memory location 000C to the output device.*

CPU

PC: 0009

IR: E1000C *11100 (E1h): character output from direct address C*

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device New character 'n' is displayed after characters 'fu'.

Bus ASCII code for 'n' is sent from memory 000C to output device.

Tracing the State of PEP/6 (13)

13) *Execution cycle continues, since no stop instruction was reached - fetch of the next instruction.*

CPU

PC: 0009

IR: 00????

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device: displays characters 'fun'.

Bus instruction from address 0009 is sent to the IR

Tracing the State of PEP/6 (14)

14) *Increment PC.*

CPU

PC: 000A

IR: 00????

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device displays characters 'fun'.

Bus no activity

Tracing the State of PEP/6 (15)

15) *Execution of instruction in IR. This is the stop instruction and cycle terminates.*

CPU

PC: 000A

IR: 00????

Main Memory

0000	E1	00	0A
0003	E1	00	0B
0006	E1	00	0C
0009	00		
000A	66		
000B	75		
000C	6E		

Output device displays characters 'fun'.

Bus no activity

II. PEP/6 OS

Operating System

- **Programming at level 3 is writing a set of instructions with appropriate data in binary**
  - To execute this, you must load program in memory
  - The operating system is responsible for this.
- **Operating System is simply a program**
  - A program that loads and executes other programs.
  - It thus resides in main memory.
- **In PEP/6, the OS resides in bottom part of memory**
  - It is simply a set of instructions that occupy memory starting location 0BE6 to 0FFF
  - The top of memory belongs to applications.

PEP/6 Operating System

- **The loader is the part of the OS program that loads applications in memory.**
- **But who loads the operating system?**
  - This is why starting up a computer is called the bootstrapping process!
  - The PEP/6 loader, along with the rest of OS is *always* present in memory.
- **This is because it resides in permanent memory called ROM.**

ROM vs. RAM

- **There are two types of electronic circuits from which memory is manufactured:**
  - Read/write elements and read-only elements.
  - The first is called *Random Access Memory (RAM)*
  - The second is called *Read-Only Memory (ROM)*
  - Like difference between CD-RW and CD-ROM
  - Or a pencil vs. pen.
- **What is important about ROM is that it cannot be changed**
  - Not even when the power is off.
  - So the operating system is permanently burned in a part of the memory implemented in ROM.

PEP/6 OS (2)

So most of PEP/6 resides in the bottom part of the memory implemented in ROM.

There is however a RAM part of the OS used to store changeable system variables.

Memory Map (1)

The top of memory is for applications. It starts with the *user* stack.

- When application programs call subroutines, this is where parameters and return values are stored
- It grows upward to lower addresses.

The OS also has a stack, located 100 bytes below the user stack at OC46.

- It also grows upward in memory
- Holds temporary OS data

Memory Map (2)

Most important parts of OS are the loader at 0C50 and the interrupt services 0C95

- The loader loads & executes programs.
- The interrupt services implement instructions by trapping 3 unimplemented instructions on the PEP/6 CPU

When you execute the loader, the system initializes the CPU SP and PC using values stored in memory at locations 0FFA-0FFC

- These are called machine vectors: pointers to start of system stack and loader program.
- Determined by computer designer and can be changed by altering ROM. Only need to change the pointers!

Loading a program (1)

- **The loader then executes, because the PC is now pointing at its first instruction in memory.**
- **The loader in PEP/6 accepts ASCII files.**
  - When it starts, it converts your ascii file to binary code.
  - You have to be very careful with the layout of your ascii source file.
- **Write a sequence in hexadecimals, like this:**

```
E1 00 07 E1 00 08 00 48 69 zz
```

  - The loader recognizes the zz as the end of the program or sentinel
  - To work correctly, first character in file needs to be hex: no spaces.
  - No trailing spaces after a line.

Loading a program (Example)

Addr.	Machine Language (hex)	
0000	E10007	;Character output
0003	E10008	;Character output
0006	00	;Stop
0007	48	;ASCII H character
0008	69	;ASCII i character

Hex Version for the Loader

```
E1 00 07 E1 00 08 00 48 69 zz
```

Output

```
Hi
```


Executing a program

- **The loader then reads you the binary code into consecutive memory locations starting at 0000.**
- **Then (after you select the execute option), the loader program re-initializes the SP and PC.**
  - It gets the location of the user stack from Mem[0FF8]=0BE6
  - Sets the stack pointer to this location.
  - It sets the program counter to 0000
  - This is location of first instruction of our application program.
- **Then, the Von Neuman Cycle begins and the CPU starts fetching consecutive instructions.**
  - Program prints “Hi” on screen.

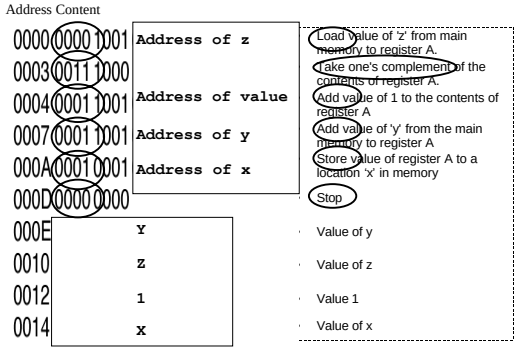
Another Couple of Examples

- **Implementing subtraction without subtract instruction**
- **By definition of the subtraction instead of x = y -z we can write x = y + (-z).**
  - The negation of a positive value can be done by combining two operations.
  - one's complement and addition of a value of 1.
- **First we will write a pseudo code for our program.**
  - Make sure that pseudo code corresponds in complexity to the machine language instructions.

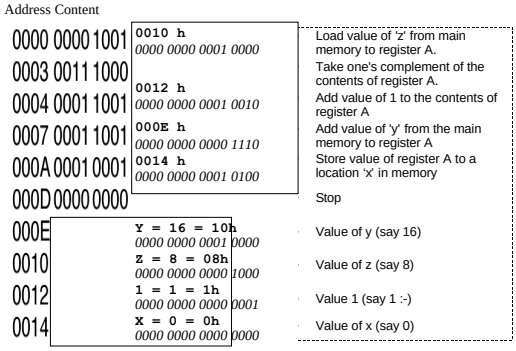
Subtract without Subtract

- Load value of 'z' from main memory to register A.
- Take one's complement of the contents of register A.
- Add value of 1 to the contents of register A (this concludes the negation of the value of 'z').
- Add value of 'y' from the main memory to register A (which now holds the negative value of z)
- Store value of register A to a location ‘x’ in memory
- Stop - this is actually quite important - without the stop the execution cycle would continue and the data following the program would be interpreted as instruction with unpredictable results!

Subtract without Subtract



Subtract without Subtract



Subtract without Subtract

Address Content

0000 0000 1001 0000 0000 0001 0000	Load value of 'z' from main memory to register A.
0003 0011 1000	Take one's complement of the contents of register A.
0004 0001 1001 0000 0000 0001 0010	Add value of 1 to the contents of register A
0007 0001 1001 0000 0000 0000 1110	Add value of 'y' from the main memory to register A
000A 0001 0001 0000 0000 0001 0100	Store value of register A to a location 'x' in memory
000D 0000 0000	Stop
000E 0000 0000 0001 0000	Value of y
0010 0000 0000 0000 1000	Value of z
0012 0000 0000 0000 0001	Value 1
0014 0000 0000 0000 0000	Value of x

Subtract without Subtract

Address Content

0000	09	00	10	• Load value of 'z' from main memory to register A.
0003	38			• Take one's complement of the contents of register A.
0004	19	00	12	• Add value of 1 to the contents of register A
0007	19	00	0E	• Add value of 'y' from the main memory to register A
000A	11	00	14	• Store value of register A to a location 'x' in memory
000D	00	00		• Stop
000E	00	10		• Value of y
0010	00	08		• Value of z
0012	00	01		• Value 1
0014	00	00		• Value of x will become 8

Subtract without Subtract: File

So this is what your ASCII text file for the loader should look like in hex machine language:

```
09 00 10 38 19 00 12 19 00 0E 11 00 14 \n
00 00 00 10 00 08 00 01 00 00 zz
```


Memory after execution

Address Content

0000 0000 1001 0000 0000 0001 0000	Load value of 'z' from main memory to register A.
0003 0011 1000	Take one's complement of the contents of register A.
0004 0001 1001 0000 0000 0001 0010	Add value of 1 to the contents of register A
0007 0001 1001 0000 0000 0000 1110	Add value of 'y' from the main memory to register A
000A 0001 0001 0000 0000 0001 0100	Store value of register A to a location 'x' in memory
000D 0000 0000	Stop
000E 0000 0000 0001 0000	Value of y
0010 0000 0000 0000 1000	Value of z
0012 0000 0000 0000 0001	Value 1
0014 0000 0000 0000 1000 (8 dec.)	Value of x = 16+ -8 = 8

Questions?