# Concordia University

## System Hardware - comp228

**Professor**: Kerly Titus **Section**: U

**Student id**: 6517722
**Student name**: Hani Sayegh
**Data submitted**: 12/02/2014

**Theory assignment** # 4
**Programming assignment** # 3

**Grade**: /100

**Department of Computer Science and Software Engineering, Fall 2014**

1)

a)

i Load 100. AC = 200, PC = 201

ii Subt 100. AC = -100, PC = 201

iii AddI 100. AC = 400, PC = 201

iv Jns 100. AC = 101, M[100] = 201, M[200] = 300, PC = 101.

b)

Load 200

| Step | RTN | PC | IR | MAR | MBR | AC |
|---|---|---|---|---|---|---|
| Initial Values | | 100 | | | | 1000 |
| Fetch | MAR <- PC | 100 | | 100 | | 1000 |
| | IR <- M[MAR] | 100 | 1200 | 100 | | 1000 |
| | PC <- PC + 1 | 101 | 1200 | 100 | | 1000 |
| Decode | MAR <- IR[11-0] | 101 | 1200 | 200 | | 1000 |
| | (Decode IR[15-12]) | 101 | 1200 | 200 | | 1000 |
| Get operand | MBR <- M[MAR] | 101 | 1200 | 200 | FFFF | 1000 |
| Execute | AC <- MBR | 101 | 1200 | 200 | FFFF | FFFF |

Add 200

| Step | RTN | PC | IR | MAR | MBR | AC |
|---|---|---|---|---|---|---|
| Initial Values | | 101 | 1200 | 200 | FFFF | FFFF |
| Fetch | MAR <- PC | 101 | 1200 | 101 | FFFF | FFFF |
| | IR <- M[MAR] | 101 | 3200 | 101 | FFFF | FFFF |
| | PC <- PC + 1 | 102 | 3200 | 101 | FFFF | FFFF |
| Decode | MAR <- IR[11-0] | 102 | 3200 | 200 | FFFF | FFFF |
| | (Decode IR[15-12]) | 102 | 3200 | 200 | FFFF | FFFF |
| Get operand | MBR <- M[MAR] | 102 | 3200 | 200 | FFFF | FFFF |
| Execute | AC <-AC + MBR | 102 | 3200 | 200 | FFFF | FFFE |

jns 300

| Step | RTN | PC | IR | MAR | MBR | AC |
|---|---|---|---|---|---|---|
| Initial Values | | 102 | 3200 | 200 | FFFF | FFFE |
| Fetch | MAR <- PC | 102 | 3200 | 102 | FFFF | FFFE |
| | IR <- M[MAR] | 102 | 0300 | 102 | FFFF | FFFE |
| | PC <- PC + 1 | 103 | 0300 | 102 | FFFF | FFFE |
| Decode | MAR <- IR[11-0] | 103 | 0300 | 300 | FFFF | FFFE |
| | (Decode IR[15-12]) | 103 | 0300 | 300 | FFFF | FFFE |
| Get operand | MBR <- PC | 103 | 0300 | 300 | 103 | FFFE |
| | M[MAR]<- MBR | 103 | 0300 | 300 | 103 | FFFE |
| | MBR<- 300 | 103 | 0300 | 300 | 300 | FFFE |
| | AC<- 1 | 103 | 0300 | 300 | 103 | 1 |
| Execute | AC <-AC + MBR | 103 | 0300 | 300 | 103 | 104 |
| | PC <-AC | 104 | 0300 | 300 | 103 | 104 |

c)

i To me it conveys that the label Data is a representaion of address 00B and addresses 002 and 003 contain references (data values) of 00B.

ii 00B, 00D, 00C, 000, 00A.

iii 200B, 8400, 000A, 004 & 006.

iv Instruction at S will be executed 10 times. We would have 10 inputs, but since we do not know what is the input we will call them a, b, c, d, e, f, g, h, i, j. So at Data we would have:
j - (i - ( h - ( g - ( f - ( e - ( d - ( c - ( b - (a - 48)))))))))

2. a)

1. Minimum number of k required is 7, because we have 100 opcodes and the smallest multiple of 2 greater than or equal to 100 is $2^7$.
2. Minimum value of n is 27, because we have $2^7$ by $2^{20}$ words, which is $2^{27}$.
3. Maximum opcodes that could be used is 128.

b)

1. Minimum n would be 33 bits, because we need at least 5 bytes to represent 27 + 7 bits as a word to the closest number of integer bytes.
2. Largest memory would be $2^{32} = 4294967296$.

3. a)

1. Total number of clock cycles is 11, since we need 4 clock cycles to load the data into AC, then we can Store it whch will take another 4 clock cycles after the AddI's are done, however since this is a pipelined processor both AddI can happen simultaneously while loading.

b)

1. We have to use the following formula, $S = \frac{nt_n}{(k+n-1)t_p}$.
2. k is 4 since we have 4 stages.
3. n is 4 since we have 4 instructions.
4. Plugging in we have: $S = \frac{16}{7}$.

c)

1. Resource conflicts.
   (a) Instructions in the pipeline may require simultaneous access to shared resources such as memory, register, bus, etc.
2. Data dependencies.
   (a) The following instructions that are in the pipeline may need data from the current instruction.
3. Conditional branch dependencies.
   (a) A branch instruction may alter the flow of execution and the instructions following the branch may not execute.

4.

a)

1.
```
MAR <- FFF    /FFF
MBR <- M[MAR] / F01
AC <- -1      / -1
AC <- AC + MBR // F00
M[MAR] <- AC /FFF has F00 now
MAR <- MBR /F01
MBR <- M[MAR] /1234
MAR <- AC /F00
M[MAR] <- MBR /F00 has 1234 now
```

2. Yes it can:

```
LoadI stackPointer  /1234 in AC

Store valueToDuplicate /1234 in valueToDuplicate
Load negativeOne / -1 in AC
Add stackPointer /F00 in AC.

Store stackPointer /F00 in stackPointer.
load valueToDuplicate / 1234 in AC
StoreI stackPointer /Address F00 now contains 1234.
```

```
stackPointer, F01 HEX /Assume stackPointer at FFF
negativeOne, -1 DEC
valueToDuplicate, 0 DEC
```

5.

a)

1. The cache has a total size of $2^9 \times 2^{10} = 2^{19}$bytes.
2. We know that each block (line) has max of 8 bytes.
3. So we have $\frac{2^{19}}{2^3} = 2^{16}$ blocks in cache.
4. Main memory has $\frac{2^9 \times 2^{20}}{2^2} = 2^{27} words.$
5. So memory has $\frac{2^{27}}{2} = 2^{26}$ blocks.
6. FOR DIRECT WE HAVE: 1 for word. 16 for block in cache. 10 for tag.
7. FOR ASSOCIATIVE: it is 1 for word and 26 for tag.
8. FOR TWO WAY SET ASSOCIATIVE: we have 1 for word, 11 for tag and 15 for set.

b)

i Progrmmed I/O should be used for transfer of large data sets.

ii Interrupt driven I/O for transfer of few bytes.

iii DMA for transfer of a few bytes within a given time limit.

6.

```
WHILE,     INPUT         /INPUT A VALUE IN ASCII
           SUBT BORDER   /SUBTRACT 48 TO CONVERT INTO DECIMAL
           SKIPCOND 000  /IF NEGATIVE HALT IMMEDIATELY, THIS IS NOT A DIGIT
           JUMP ELSE
           JUMP ENDLOOP
ELSE,      SKIPCOND 400 /SKIP IF INPUT IS 0
           JUMP ISDIGIT
           JUMP INCREMENT
ISDIGIT,   SUBT MAX /SUBTRACT 9 TO SEE IF IT IS DIGIT
           SKIPCOND 800 /SKIP IF MORE THAN 0,
                       / THIS MEANS IT IS NOT A DIGIT SO TERMINATE
           JUMP WHILE   /WHILE WE HAVE INPUT
           JUMP ENDLOOP
```

```
INCREMENT, LOAD COUNT
           ADD FOUND  /INCREMENT NUMBER OF 0'S BY 1
           STORE COUNT
           JUMP WHILE /WHILE WE HAVE INPUT

ENDLOOP, LOAD COUNT
         OUTPUT
         HALT


FOUND,  DEC 1 /ADD 1 EVERYTIME WE FIND A 0
BORDER, DEC 48 /SUBTRACT48 TO CONVERT FROM ASCII TO DECIMAL
MAX,    DEC 9 /IF WE SUBTRACT 9 FROM DECIMAL AND IT
              /IS GRETER THAN 0 THEN IT IS NOT A DIGIT
COUNT, DEC 0 /VARIABLE THAT CONTAINS NUMBER OF 0'S
```

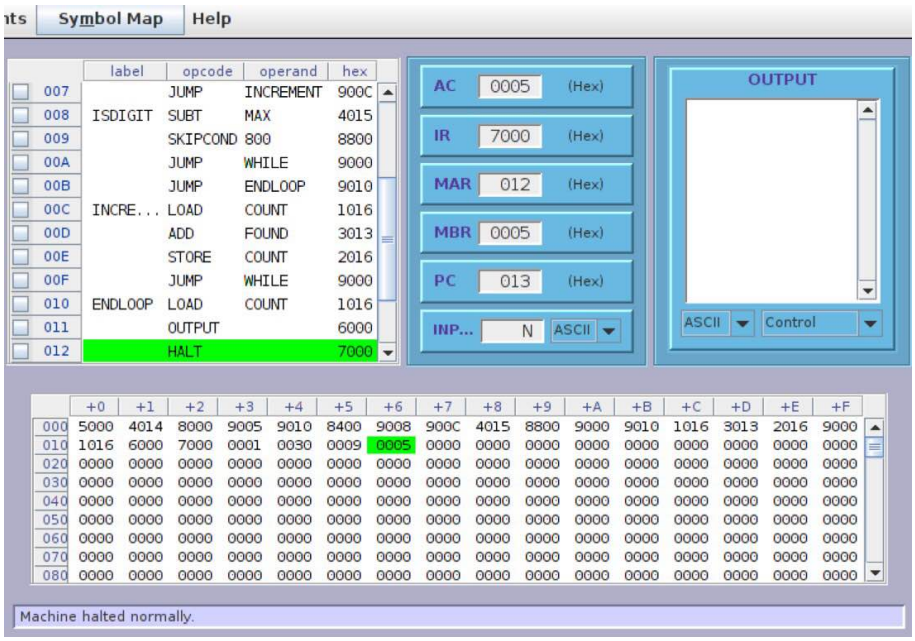


Figure 1: A simulation with five 0's occurring and halted by ASCII key N

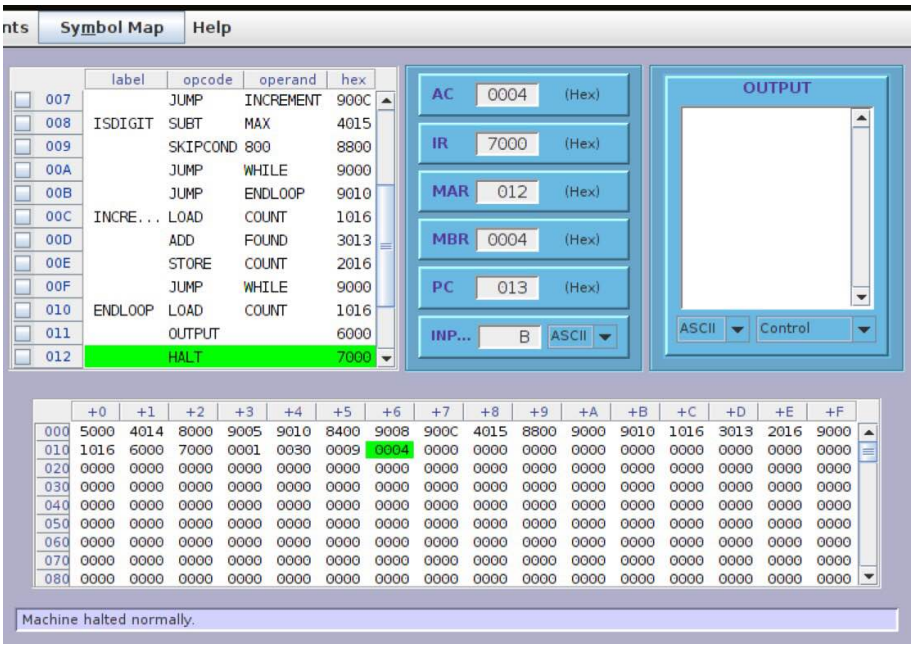


Figure 2: A simulation with four 0's occurring and halted by ASCII key B