MONASH University

FIT1047 Applied Session Week 5

ASSEMBLY LANGUAGE PROGRAMS (USING MARIE)-ADVANCE

OBJECTIVES

- The purpose of this applied session is getting to know subroutines and indirect addressing in MARIE assembly code programs

INSTRUCTIONS

- Use MARIE simulator to write assembly language programs.
- You may work in a small group.

Activity 1: MARIE Subroutines

An important concept in programming is that of a procedure, a function, or a subroutine, a piece of code that has a fixed purpose and that needs to be executed over and over again. Most ISA (Instruction Set Architectures) have some level of support for writing subroutines. In MARIE, there's the JnS X instruction ("jump and store"): It stores the value of the PC at memory address X and then jumps to address X+1. The value stored at X is called the return address, i.e., the address where execution should continue once the subroutine has finished its job. To return from a subroutine, the last instruction in the subroutine should be a jump back to the return address. This can be achieved using the JumpI X instruction: it jumps to the address stored at address X (compare that to Jump X which jumps to the address X).

Write a MARIE program to read a number from memory, double it and display the new number. Then convert it to a subroutine, and create a main program to accept any number from the user (via keyboard), if it is "zero" exit the program, otherwise call the subroutine to double it and display.

Sample Solution:

```
/Subroutine Double & Main (Exits when user inputs '0'
Dbegin,     Input       /input and store in numIn???
            Skipcond    400  /check for Zero
            Jump        Lcont
            Jump        Lend
Lcont,      Store       numIn
            Output
            Load        numIn
            Store       numTemp          / passing the number to sub
```

```
                JnS         subDouble       /sub call
                Load        numTemp
                Store       numRes          /getting the result from the sub
                Output
                Jump        Dbegin
Lend,           Halt
/main ends here
/sub begins here
subDouble,      HEX 0       /subroutine name
                Load        numTemp         /double
                Add         numTemp
                Store       numTemp
                JumpI       subDouble       /last line of a sub
/sub ends here

/variables
numIn,      DEC 0 / store a number input from KB
numRes,     DEC 0 / store a double of any number
numTemp,    DEC 0
```

Activity 2: Multiplication Operations in MARIE

MARIE has arithmetic instructions only for adding and subtracting numbers, but not for multiplication. Your task is to implement a multiplication algorithm. You can use the following pseudo-code as a starting point:

```
01 C := 0
02 if A = 0 then go to line 06
03 C := C + B
04 A := A - 1
05 go to line 02
06 halt
```

The algorithm used is based on the fact that multiplication of two numbers A and B (that is A * B) is actually summing up B, A times. (i. e. B+ B + ... A times). The algorithm first tests whether A is zero, in which case the answer is zero. Otherwise, keep looping until B is summed up A times. It assumes that variables (or storage locations) A and B hold the two integers that should be multiplied, and the result should be stored in C.

(a) Implement the algorithm using the MARIE simulator. Test your code with several different inputs.
(b) Then convert the multiplication program into a subroutine, and create a main program to accept two numbers from the user (via keyboard), if any of them is “zero” exit the program, otherwise call the subroutine to perform multiplication operation and display the result.

Sample Solution:

(a) Multiplication Program

```
                Load   Zero          / AC = Zero
                Store  numC          / numC = AC
```

```
Lbegin,     Load   numA        /02 if A = 0 (or postv) then jump to line 06
            Skipcond 800       /check for AC to be postv or not
            Jump   mDone
            Load   numC        /AC = numC
            Add    numB        /AC = AC + numB
            Store  numC        /numC = AC
            Load   numA        /AC = numA
            Subt   One                 /AC = AC - One
            Store  numA        /numA = AC
            Jump   Lbegin      /jump to Lbegin
mDone,      Load   numC
            Output
            Halt               /halt
/variables
numA,       DEC 4
numB,       DEC 5
numC,       DEC 0
One,        DEC 1
Zero,       DEC 0
```

(b) Multiplication subroutine and main program

```
/main program begins here
mBegin,     Input
            Store  n1st
            Skipcond  400  /check for Zero
            Jump   Lcont1
            Jump   mEnd
Lcont1,     Input
            Store  n2nd
            Skipcond 400  /check for Zero
            Jump   Lcont2
            Jump   mEnd
Lcont2,     Load   n1st
            Store  numA
            output
            Load   n2nd
            Store  numB
            output
            JnS    subMult
            Load   numC
            Output
            Jump  mBegin
mEnd,       Halt
/main program ends here
/variables
n1st,       DEC 0
n2nd,       DEC 0
nRes,       DEC 0
/subroutine begins here
subMult,    HEX 0
            Load   Zero         / AC = Zero
            Store  numC         / numC = AC
Lbegin,     Load   numA         /02 if A = 0 (or postv) then jump to line 06
            Skipcond 800        /check for AC to be postv or not
            Jump  mDone
            Load   numC         /AC = numC
            Add    numB         /AC = AC + numB
            Store  numC         /numC = AC
            Load   numA         /AC = numA
            Subt   One                  /AC = AC - One
            Store  numA         /numA = AC
            Jump  Lbegin        /jump to Lbegin
mDone,      JumpI subMult
/variables
numA,       DEC 0
numB,       DEC 0
numC,       DEC 0
```

```
One,            DEC 1
Zero,           DEC 0
/subroutine ends here
```

Activity 3: Indirect Addressing in MARIE

In MARIE, we can make use of the instructions *"LoadI*" and "*StoreI*" (handling indirect addressing) to store and retrieve data from MARIE memory.

```
        ……..
        StoreI  MemAdd
        LoadI   MemAdd
        ……..
MemAdd, HEX 020
```

Referring to the example code above, *StoreI* command will store the content of AC into the memory location "HEX 020," and *LoadI* command will load the content of the memory location "HEX 020" into the AC.

(a) Write a MARIE program to input decimal numbers from the keyboard and store them starting from memory location "HEX 020." The program should terminate when the user enters a number '0'. Note: one decimal number (e.g. 20) to be stored in one memory location.
(b) Extend your program to load and display the numbers (starting from memory location HEX 020). Continuing from part (a), once the user enters a number '0', your program should terminate the number entry and load these numbers from memory to display one after the other in the output window.

Sample Solution:

(a)

```
        Load    MemAdd
        Store   tempAdd
begin,  Input
        Store   tempIn
        Skipcond  400  /check for Zero
        Jump    sCont
        Jump    sEnd
sCont,  Load    tempIn
        StoreI  tempAdd
        Load    tempAdd
        Add     One
        Store   tempAdd
        Jump    begin
sEnd,   Halt
tempIn,         DEC 0
MemAdd,         HEX 020
tempAdd,        HEX 0
One,            DEC 1
```

(b)

```
        Load    MemAdd
        Store   tempAdd
```

```
begIn,  Input
        Store   tempIn
        Skipcond  400  /check for Zero
        Jump   sCont1
        Jump   sDisp
sCont1,Load   tempIn
        StoreI  tempAdd
        Load   tempAdd
        Add    One
        Store   tempAdd
        Jump   begIn
sDisp,  Load   MemAdd
        Store   tempAdd
begOut,LoadI tempAdd
        Store   tempIn
        Skipcond  400  /check for Zero
        Jump   sCont2
        Jump   sEnd
sCont2,Load   tempIn
        Output
        Load   tempAdd
        Add    One
        Store   tempAdd
        Jump   begOut
sEnd,  Halt
tempIn,         DEC 0
MemAdd,         HEX 020
tempAdd,        HEX 0
One,            DEC 1
```

Activity 4: Handling String (using Indirect Addressing) in MARIE

In MARIE assembly language programming, we can make use of the ADR command, the HEX keyword and a label “myName” to store a string in memory (example below).

```
myAdd,      ADR   myName
myName,     HEX 04A     /’J’
            HEX 06F     /’o’
            HEX 068     /’h’
            HEX 06E     /’n’
            HEX 04E     /’N’
            HEX 06F     /’o’
            HEX 061     /’a’
            HEX 068     /’h’
            HEX 0       /
```

Here, the label “myAdd” refers to the memory location of the label “myName”. So, “myAdd=0001” refers to the first character ‘J.’

Instr. No.	Memory Location	Label	Code	Memory Contents
1	000	myAdd,	ADR myName	0001
2	001	myName,	HEX 04A /J	004A
	002		HEX 06F /o	006F
	003		HEX 068 /h	0068
	004		HEX 06E /n	006E
	005		HEX 04E /N	004E
	006		HEX 06F //o	006F
	007		HEX 061 //a	0061
	008		HEX 068 //h	0068
	009		HEX 0 //.	0000
	00A			

Figure 1

Write a MARIE subroutine to load the characters (into the AC) one-by-one using indirect addressing (LoadI) and modify the character by manipulating the ASCII code (adding ‘1’ to the ASCII value) and store the character back into the same memory location using StoreI command. Verify the changed ASCII values in the MARIE memory.

Sample Solution:

```
/string manipulation begins here
        Load    myAdd
        Store   MemAdd
begin,  LoadI   MemAdd
        output
        Store   tempIn
        Skipcond  400  /check for Zero
        Jump    sCont
        Jump    sEnd
sCont,  Load    tempIn
        Add     One
        StoreI  MemAdd
        output
        Load    MemAdd
        Add     One
        Store   MemAdd
        Jump    begin
sEnd,   Halt
tempIn,         DEC 0
MemAdd,         HEX 0
One,            DEC 1
myAdd,          ADR   myName
myName,         HEX 04A       /'J'
                HEX 06F       /'o'
                HEX 068       /'h'
                HEX 06E       /'n'
                HEX 04E       /'N'
                HEX 06F       /'o'
                HEX 061       /'a'
                HEX 068       /'h'
                HEX 0         /
```