ICT114 Supplementary Reading Materials 3:

Assembly Language for Motorola 68000 based Microprocessor

Edited by Tan Sim Bee and Wong Yoke Moon

Introduction

A course in computer system architecture is incomplete without looking at a processor's instruction set, its addressing modes and some programming examples. It is the "link" between hardware and software. For this course, the Motorola 68000 processor is chosen because of its classic architecture. It is comparatively easier to understand compared to the Pentium processor family. I shall use a ***simplified model*** of the 68000 with a simplified instruction set to introduce these concepts as this course in not about assembly language programming.

What is the different?

- **Real 68000 or 68K processor**
  8 data registers (D0-D7)
  8 address registers (A0-A7)
  Status Register, 16-bit with 12 valid bits
  Very complex set of instructions
  Many addressing modes

- Our ***Simplified 68K*** or **E114 processor as we shall named it.**
  8 data registers (D0-D7) -- same as 68K
  8 address registers (A0-A7) -- same as 68K (32 bits)
  Status Register, 16-bit with 2 valid bits, Zero and Negative flags
  A subset (only 15 of instructions) of the 68K instruction set
  Only four addressing modes

E114 Registers

No data or address register is different from any other. That means whenever you can use D0, you can always use D2 - D7. Similarly, for address registers A0 to A7. A7 is also being used as the processor's stack pointer for stack operation.

32-bit Data Registers

One important concept is that if you have a 32-bit word (also called a longword) stored in a data register and you move a byte i.e. an 8-bit word, into it, the most significant three bytes are still there! This may cause problems when you later add or subtract the register, you MUST clear the most significant part!

E114 processor Program Counter

The E114 processor also has a PC, Program Counter. It is a 24-bit.

Status Register or Flag Register

The E114 processor has a 16-bit register called the status register. The bits are set or cleared by the results of various operations. In this context, *set* means equal to 1, and *cleared* means equal to 0. At times the bits are set and cleared, or *conditioned,* automatically by the processor itself; other times they are conditioned by specific program instructions.

Programs can check these bits and use the results of the check for whatever purpose the programmer has in mind, often to decide on a branch. The bits are sometimes called flags

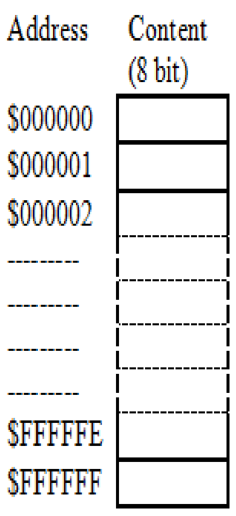
There are usually a couple of flags in the status register. For the E114 processor we shall only consider two flags, namely zero or Z flags and negative or N flag.

Z-flag (Zero).
Is set if the result of an operation is zero.

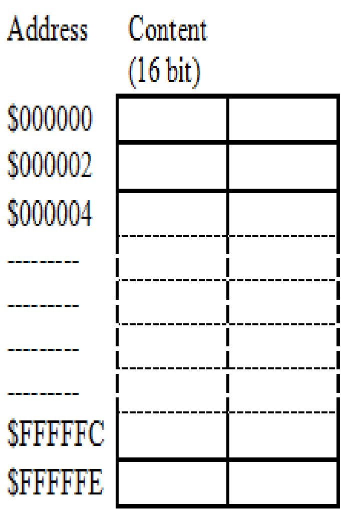
N-flag (Negative).
If the highest bit in the result is set (in two complement it means the sign bit), N will be set.

E114 Processor memory space

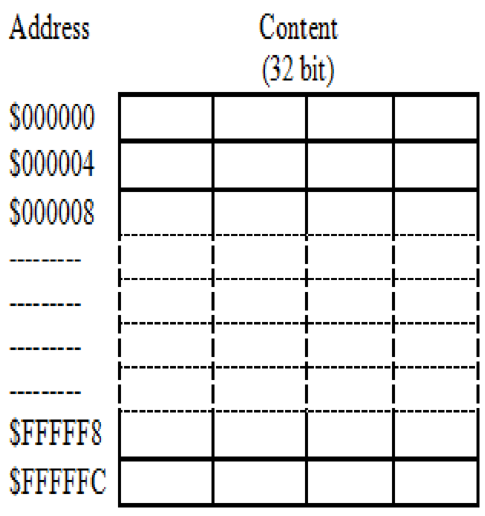
The E114 memory space is byte-addressable. Successive bytes are stored at consecutive byte addresses $000000, $000001, $000002, $000003 …. . Successive 16-bit words are stored at consecutive even addresses $000000, $000002, $000004, $000006 …., and 32-bit longword are stored at consecutive addresses $00000, $000004, $000008, $00000C …..



a) byte addressing



b) word addressing



c) longword addressing

If a 32-bit longword $12345678 is stored in at address $001000, this is how it is stored, see figure below.

Address	Content (8 bit)
$001000	$12
$001001	$34
$001002	$56
$001003	$78
$001004	

$FFFFFE	
$FFFFFF	

Address	Content (16 bit)	
$001000	$12	$34
$001002	$56	$78
$001004		

$FFFFFC		
$FFFFFE		

Address	Content (32 bit)			
$001000	$12	$34	$56	$78
$001004				
$001008				

$FFFFF8				
$FFFFFC				

E114 processor instructions

All processors have built into them a set of simple instructions they can operate - Processor's instruction set. Different processors have different instruction sets. The E114 processor has 15 instructions that can be divided into four categories:

Data movement

MOVE

This instruction allows data transfers from register to register, register to memory, memory to register, and memory to memory. Typically, byte, word, or long-word data can be transferred.

Data transformation

ADD, SUB,
NOT, AND, OR

These are the logical and arithmetic instructions

Sequence control

BNE, BMI, BPL, BRA, JMP
BSR, and RTS
NOP - do nothing instruction
STOP - stop the program

These instructions do not specify the operands. Operands are predefined.

These 15 instructions are all that ***you are required to know*** for this course.

Byte, word and longword instructions

Most instructions of the E114 processor have a suffix that shows how much data should be used in the instructions. This can be a byte (.B), a word (.W) or a longword (.L).
Example:
MOVE.W D0, D1 moves (or copies) the first 16 bits (a word) from D0 to D1.

E114 processor Addressing Mode

Most instructions have two to three fields.
Example:
Operation code + operand
Operation code + source operand + destination operand

The address (i.e. where) of operand can be specified in many ways, collectively called the addressing modes.

Note:
xxxx is a decimal number.
$xxxx is a hexadecimal number (without $ number is decimal)
%yyyyyyyy is a binary number

1. Immdeicate addressing (source operand is a Data)

Actual operand is part of the instruction
e.g. MOVE.L #$12345678, D0 (Data $12345678 is loaded to register D0)

The operand $12345678 is part of the instruction

2. Direct or absolute addressing (source operand is an Address)

Operand's address is part of the instruction
e.g. MOVE.L $1234, D0 (contents at address $1234 is loaded to register D0)

$1234 is the address of the operand

3. Register direct addressing

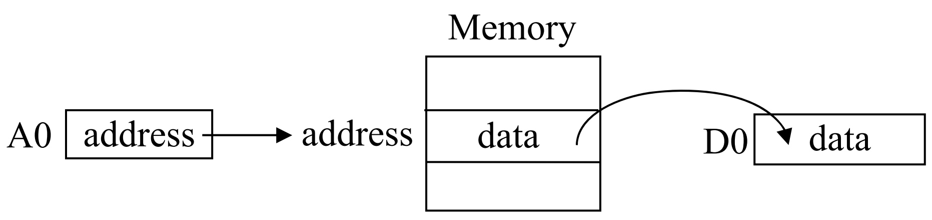
Operands (source and destination) are the content of the registers
e.g. MOVE.L D1, D0

D0 content is the operand.

4. Address Register indirect addressing

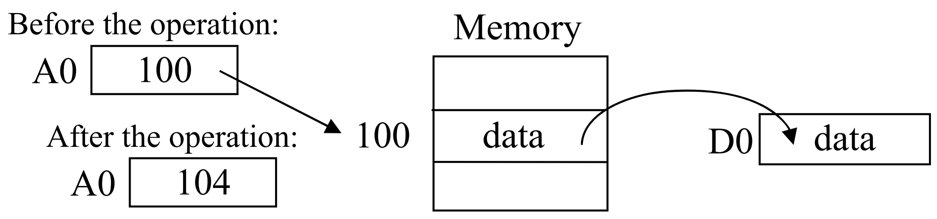
Operand's address is in an address register
e.g. MOVE.L (A0), D0 (Contents at addr given in A0 is loaded to D0)

Register A0 content is the address of the operand



5. Address Register indirect with Post-Increment addressing

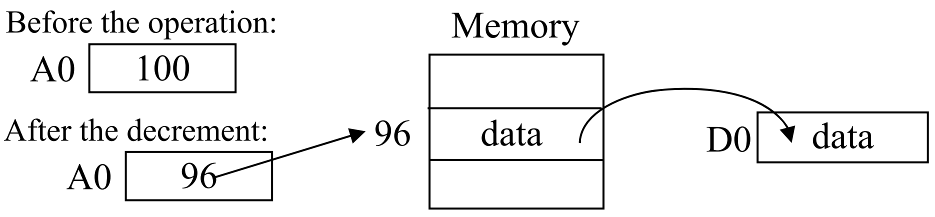
Same as 4, except that the address register content is increment by 1 for .B, 2 for .W or 4 for .L *after* the operation
e.g. MOVE.L (A0)+, D0



6. Address Register indirect with Pre-decrement addressing

Same as 4, except that the address register content is decrement by 1 for .B, 2 for .W or 4 for .L ***before*** the operation

e.g. MOVE.L -(A0),D0



7. Relative addressing

Conditional Branch

e.g. BEQ <value> (This instruction is not in E114 Instruction set)

PC value (start of next location) is only updated i.e. PC= PC + <value> when the condition specify by the instruction is met.

Unconditional Branch

e.g. BRA <value>

PC value (start of next location) is always updated. PC = PC + <value>

8. Absolute addressing

e.g. JMP <value> PC value is replaced by <value>

Note: Normally a label is used instead of the actual number i.e. <value>. The assembler (a code conversion program will convert it to <value>).

Summary of the E114 instructions is given in **appendix 1**

Structure of the E114 processor program

Below is a typical assembly language program i.e. a program written in E114 processor's instructions.

```
      Label     Instruction    Operand          Comment
1               ORG            $1000            ; program code loaded at address $001000
2     START     MOVE.W         #$2345, D0
3               ADD.W          DAT, D0          ; sum is in D0
4               MOVE.W         D0, RESULT       ; store sum in location RESULT
5               STOP           #$2700           ; program stops
6
7               ORG            $2000
8     DAT       DC.W           $5678            ; item stored at DAT is initialized to $5678
9     RESULT    DC.W           0                ; item stored at RESULT initialized to 0
10              END            START
```

Important Note:
The line numbering added in the first column is not part of the program but is used to identify the instructions.

The program above is divided in four columns, namely labels, instructions, operands and comments. The heading added is not part of the program.

1. START, DAT and RESULT in *lines 2, 8 and 9* are labels. Labels provide alternative for numeric addresses. The assembler will translate labels to actual numeric addresses.

2. ORG, DC.W and END are assembler directives. They are control statements used by the assembler.

   ORG means origin, tells the assembler where the instructions or data are to be loaded in memory. ORG in *line 1* tells the assembler to load the following instructions starting at address $1000. ORG in *line 7* tells the assembler to load the following data starting at address $2000. DAT and RESULT are labels with address $2000 and $2002 respectively ($5678 is stored in 2 memory locations).

   DC.W in *line 8* means *define constant* places the word $5678 in memory and labels it as DAT. *DC.W 0* in *line 9* places 0 in memory and label it as RESULT.

   END is a directive telling the assembler that is the end of the program.

3. Column 4 is for putting comments. All comments are prefixed by semicolon.

4. STOP #$2700 places #$2700 into the status register. #$2700 halts the processor until an interrupt. ***We shall use STOP #$2700 for all our programs in this course.***

Stack and Subroutine

- Reduce repetition of a common block of codes
- Break a program into smaller and manageable parts
- Save memory space
- A ***BSR*** instruction causes a subroutine to be executed.
- A ***RTS*** instruction causes a return from subroutine to the main program.

Question: How does the processor know where to return?
Answer: Stack keeps the return address.

In the example program, when ***BSR PUTS*** is executed, the address of the next instruction **BRA START** is put on the stack. The program then branches to the subroutine ***PUTS.*** When the last instruction of the subroutine ***RTS*** is executed, the program counter is loaded with the return address and the program resumes at instruction ***BRA START.***

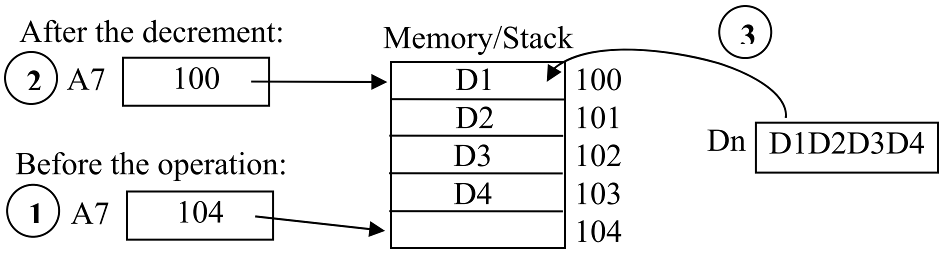
```
        ORG   $1000
START         ......        ; START OF MAIN PROGRAM
              ......
        BSR   PUTS          ; BRANCH TO SUBROUTINE "PUTS"
        BRA   START         ; KEEP LOOPING!

        ORG   $3000
PUTS          .....         ; STARTS OF SUBROUTINE
              .....
              .....
        RTS

        END   START
```

The E114 processor STACK:
By convention the E114 processor uses a stack to support subroutines. The stack is implemented in memory with address register A7 pointing to the top of the stack (also called a stack pointer, SP). That is, A7 contains a value which corresponds to the *address* of the top element of the stack in memory. The E114 stack grows downwards. The stack starts at a ***high address*** and grows towards ***smaller addresses***.

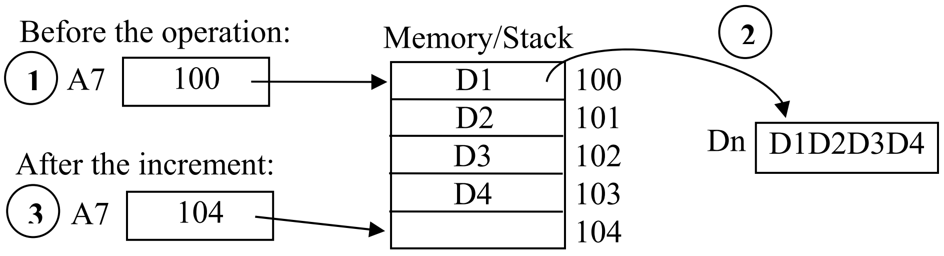
In the E114 processor there is no way to represent an empty stack. The A7 always has a value and this always corresponds to a valid address. If you do use A7 to read from memory you will read the element at the top of the stack. If you did not push an element you are just reading the value that happens to be there where A7 points to. It would be garbage for all we care but you will still be able to read a value. So, whether the stack is empty and how elements it holds is a concept that only the programmer can understand, and it is all a matter of convention and of using the appropriate operations in sequence to achieve the effect we desire.

E114 Stack operation

MOVE.L Dn, (A7)
- Push operation: Dn data store on the stack. A7-4 =A7 after operation



MOVE.L (A7)+, Dn
- Pop operation: Stack data is put back to Dn. A7+4 =A7 after operation



Assembler, complier and loader
Translation Software is called compiler or assembler. Compliers are used to translate high level languages like Basic, Java, C++ etc. An assembler translates the assembly language program to machine codes. Another program (called a loader) loads the machine codes into primary memory for the processor to execute. Processor can only execute codes that are stored in main or primary memory

Reference
Principles of Computer Hardware ***Alan Clements***

Appendix 1

E114 Processor Instruction Set Summary

Operation	Flags NZ	Length BWL	Description	Remarks
Move				
MOVE <EA1>,<EA2>	**	XXX	move data	
Arithmetic				
ADD <EA1>, An or Dn	**	XXX	add binary	
SUB <EA1>, An or Dn	--	XXX	subtract binary	An - <EA1>
Logical				
AND <EA3>, Dn	**	XXX	logical AND	
NOP <EA3>, Dn	--		no operation	
OR <EA3>, Dn	**	XXX	logical OR	
NOT Dn	**	XXX	logical NOT	
Control				
BRA <Label>	--		branch always	
BSR <Label>	--		branch subroutine	
BNE <Label>	-*		branch not equal to 0	
BMI <Label>	*-		branch on negative	
BPL <Label>	*-		branch on positive	
JMP <Label>	--		jump always	
RTS	--		return from subroutine	
STOP #$2777			stop	

$ - hexa
# - immediate

<EA1> = Dn, An, (An), (An)+, $<data>, #$<data>
<EA2> = same as <EA1> except #$<data>
<EA3> = Dn, #<data>

An:	**Address register A0 to A7**
Dn:	**Data register D0 to D7**
(An) :	**Content of address whose address is in An**
(An)+:	**Similar to (An) but after instruction An=An+1 for.B, An+2 for .W and An+4 for .L**

$nnnn:	**16-bit Hexadecimal address**
$nnnnnnnn:	**32-bit Hexadecimal address**
#$nn:	**8-bit immediate Hexadecimal data**
#$nnnn:	**16-bit immediate Hexadecimal data**
#$nnnnnnnn:	**32-bit immediate Hexadecimal data**