The 68000

Lesson 7 – Code Production

Introduction to the assembly

- We want to understand how to get an executable program from an assembly code.
- Example:

```
        DATA
X:      DC.L   1
        CODE
        MOVE.L #3,D4
        ADD.L X,D4
        RTS
```

Example (cont'd)

- The previous code is transformed in a series of binary codes:
  - $0000 |
    $000C | HEADER
    $0000 |
    $0004 |
    $283C | |
    $0000 | |
    $0003 | | instructions codes
    $D8AD | |
    $0000 | |
    $4E75 | |
    $0000 | | | data code
    $0001 | | |

What's the point?

- The point of this chapter is to understand how we get this code sequence.
- The first thing we can see is that there is a header containing the size of the instructions zone and the data zone.
- To begin the execution, the instructions codes are placed somewhere in memory write-protected. Data is stored in another location.
- The execution will also need to access the stack.

Pre-treatment

- The first thing to do is take off every comment, constants and macros (when used)
- Look the following example...

Pre-treatment: example

Before pre-treatment	After pre-treatment
MAX EQU 10 ... MOVE.L #MAX,D2	MOVE.L #10,D2
MACRO EXCHANGE(X,Y) MOVE.L X,D0 MOVE.L Y,X MOVE.L D0,X ENDMACRO ... EXCHANGE A2,A3	MOVE.L A2,D0 MOVE.L A3,A2 MOVE.L D0,A2

Coding an instruction

- Each instruction element is coded with a sequence of bits. Putting all these together we get a word. The instruction elements are:
  - The operation (MOVE, ADD, ...)
  - The size (B, W, L)
  - Source address (#data, Dn, ...)
  - Destination address (Dn, d(An), ...)
- The operation is generally coded in variable number of bits (2 to 16), the size is generally coded with 2 bits, addresses are generally in 6 bits plus eventually one or to supplementary words.

How to code an address

- When using a register (direct, indirect w/o shifting), 3 bits are used to know the addressing mode and 3 other to tell the register number.
- For indirect addressing with shifting "d(An)", we also use 3 bits to show the addressing mode and 3 other to show the number of the registry. We use another word to code the shifting.
- For absolute addressing, we use 6 bits to indicate the addressing mode and a longword to code the actual address.
- For immediate values, we use 6 bits to show the addressing mode plus one or two words to code the value. If the size is B, we use a word and store the number in bits 0 to 7. If it’s a W we use a word. If it’s a L, we use 2 words.

Coding MOVE

CODAGE DE MOVE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Taille		Destination Registre			Mode			Source Mode			Registre		

Taille	
B	01
W	11
L	10

Mode d'adressage	Mode	Registre
Dn	000	numéro
An		
(An)	010	numéro
(An)+	011	numéro
-(An)	100	numéro
d(An)	101	numéro

Mode d'adressage	Mode	Registre
d(An,Xi)	110	numéro
Abs.W	111	000
Abs.L	111	001
d(PC)		
d(PC,Xi)		
Imm		

Destination

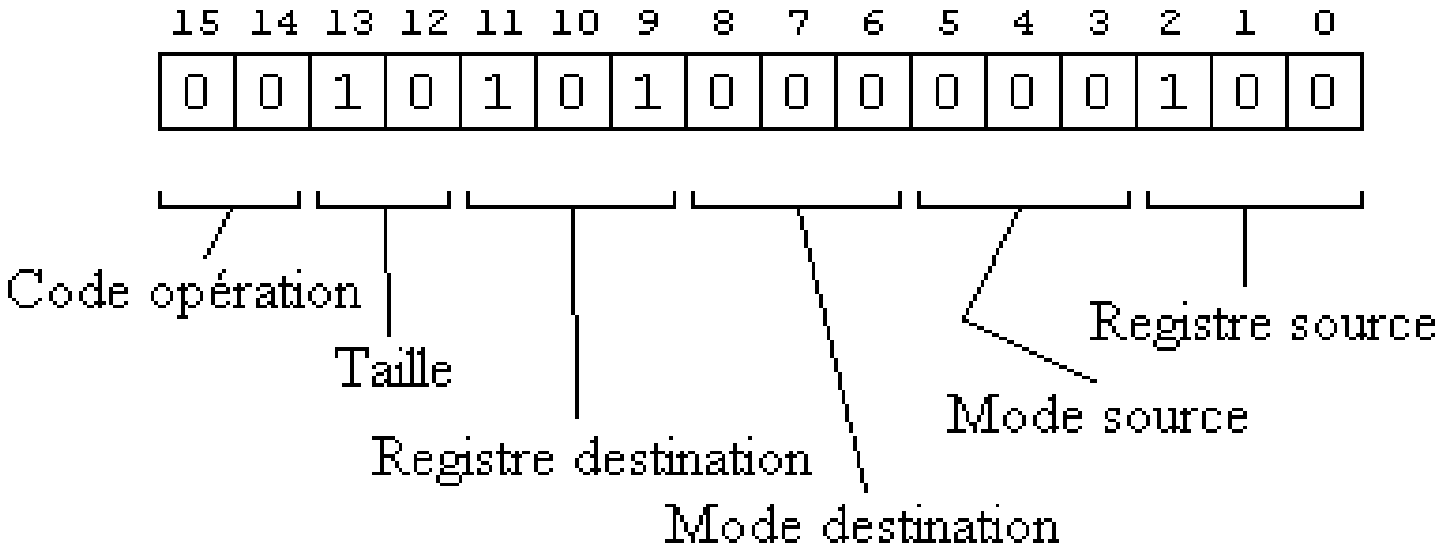
Mode d'adressage	Mode	Registre
Dn	000	numéro
An	001	numéro
(An)	010	numéro
(An)+	011	numéro
-(An)	100	numéro
d(An)	101	numéro

Mode d'adressage	Mode	Registre
d(An,Xi)	110	numéro
Abs.W	111	000
Abs.L	111	001
d(PC)	111	010
d(PC,Xi)	111	011
Imm	111	100

Source

MOVE – an example

- MOVE.L D4, D5



Coding ADD – 1st version

ADD <ea>,Dn

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	1	Registre	Taille	Source Mode | Registre

Taille	
B	000
W	001
L	010

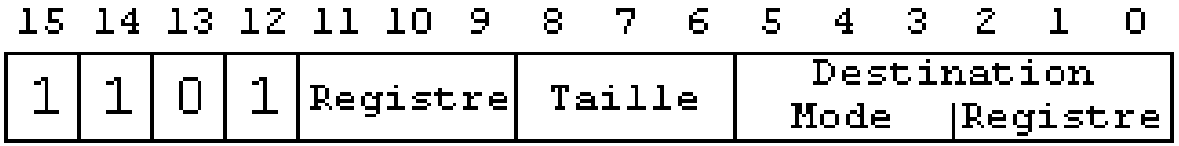
Mode d'adressage	Mode	Registre
Dn	000	numéro
An	001	numéro
(An)	010	numéro
(An)+	011	numéro
-(An)	100	numéro
d(An)	101	numéro

Mode d'adressage	Mode	Registre
d(An,Xi)	110	numéro
Abs.W	111	000
Abs.L	111	001
d(PC)	111	010
d(PC,Xi)	111	011
Imm	111	100

Source

Coding ADD – 2nd version

ADD Dn,<ea>



Taille	
B	100
W	101
L	110

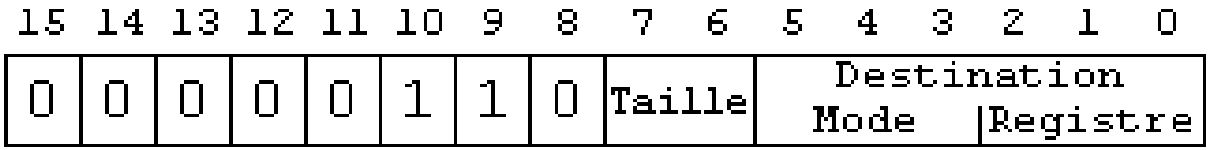
Mode d'adressage	Mode	Registre
Dn		
An		
(An)	010	numéro
(An)+	011	numéro
-(An)	100	numéro
d(An)	101	numéro

Mode d'adressage	Mode	Registre
d(An,Xi)	110	numéro
Abs.W	111	000
Abs.L	111	001
d(PC)		
d(PC,Xi)		
Imm		

Destination

Coding ADD - 3rd version

ADDI #data,<ea>



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Taille		Destination Mode			Registre		

Taille	
B	00
W	01
L	10

Mode d'adressage	Mode	Registre
Dn	000	numéro
An		
(An)	010	numéro
(An)+	011	numéro
-(An)	100	numéro
d(An)	101	numéro

Mode d'adressage	Mode	Registre
d(An,Xi)	110	numéro
Abs.W	111	000
Abs.L	111	001
d(PC)		
d(PC,Xi)		
Imm		

Destination

Little branching trouble

- Let's see an example
  - ```
            move.l #1,D1
            bra next
            add.l #3,D1
    next:   rts
    ```
- When coding line by line we cannot know where "next" is while coding "BRA NEXT"
    ```

Object and executable code

- When parsing the lines the first time we cannot produce the complete code for BRA NEXT, though we can something that should be modified later on: it's the basics of 2 steps assembling (code production + code modification)
- It's the assembly and link-editing sections
- After first phase we get an object code
- After the second phase we get the executable code

Example

- In the previous BRA NEXT
  - We produce $6000 (code of BRA) and $0000 (temporary move for the PC).
  - We memorize that this entry should be modified later on
  - We use a symbol table containing a certain amount of double entries (name of reference – address of the word to change)

2 phase assembly: Phase 1

- For each source line:
  - If a label is defined, memorize the label and associated address in a table “tlab”
  - Create the instruction code. If a reference is present in the instruction, use the value 0 for its coding and save the reference in a table “tref”
- Save the results in a file holding a Header, the produced code, the “tlab” table and the “tref” table

2 phase assembly: Phase 2

- For each element of the “tref” table
  - Modify the corresponding code checking the address given in the “tlab” table
- Save to produced code in a file. This executable file holds a header, the code and eventually, the “tlab” and “tref” tables if there’s a possibility to dynamically edit the links.

Coding BRA

- BRA <label>
- Other branching

BRA <étiquette>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Mot 1
Déplacement codé sur 16 bits																Mot 2

Branchements conditionnels
Bits 11 à 8

CC	0100	LS	0011
CS	0101	LT	1101
EQ	0111	MI	1011
GE	1100	NE	0110
GT	1110	PL	1010
HI	0010	VC	1000
LE	1111	VS	1001

Loading programs – The loader

- The loader loads in memory the codes in the executable file and launches the program execution. It starts decoding the Header of the file in order to find the instructions codes and data codes. Then it:
  - Reserves a memory zone to copy the instructions codes
  - Reserves a memory zone to copy the data codes
  - Launches the execution

Some interesting facts

- In a simplified machine, the execution can be triggered with the instruction “JSR (A0)”, after having the loader put the first instruction address into A0. In a more complex machine, the program memory zone has to be protected.
- The program will execute correctly if the addresses and moves created during the link-editing are compatible with the addresses of the memory zones used for the execution (this happens when the link-editor knows how the loader works)