Lecture 3: MC68000 instruction set

- **Assembler directives (the most important ones)**
  - ORG, EQU, END, DC, DS, EXTERN/PUBLIC
- **Instructions (the most important ones)**
  - Data movement
  - Integer arithmetic
  - Boolean
  - Shift and rotate
  - Bit manipulation
  - Binary Coded Decimal
  - Program flow
  - System control

Assembler directives

Assembler directives

- **are** instructions to the assembler program
  - and they appear in the mnemonic (opcode) field of the source code
- **are not** instructions to the microprocessor
  - and they have no direct effect on the contents of memory (except DC)

They cover a number of functions, including

- defining symbols and assigning them values
- controlling the flow of execution of the assembler
- setting format and content of the object and listing files

Assembler directives (the most important ones)

DIRECTIVE	OPERATION	SYNTAX		
ORG	set program origin		`ORG`	`value`
EQU	equate value to symbol	`symbol`	`EQU`	`value`
END	end of source program		`END`	`label`
DC	define data constant	`[label]`	`DC`	`number[,number][…]`
DS	define RAM storage	`[label]`	`DS`	`count`
RSEG	begin relocatable segment		`RSEG`	`name`
EXTERN	define external symbol		`EXTERN`	`symbol[,symbol][…]`
PUBLIC	define public symbol		`PUBLIC`	`symbol[,symbol][…]`

The ORG directive

- **FUNCTION (ORIGIN)**
  - Sets the starting address in memory for the instructions or data constants that follow
- **EXAMPLE**

```
00001000                        1    ORG       $1000
00001000  203C 00000012         2    MOVE.L    #$12,d0
```

- **NOTES**
  - Hex address $1000 is set as the starting address for the following instruction
  - The opcode for MOVE.L goes in address $1000
    - The second word for MOVE.L goes in address $1002 … and so on

The EQU directive

- **FUNCTION (EQUATE)**
  - Assigns a value to a symbol. The symbol is used later in the program in place of the value
- **EXAMPLE**

```
00001000                          1          ORG      $1000
00000100                          2  count   EQU      $100
                                  3
00002000                          4          ORG      $2000
00002000  203C 00000100           5          MOVE.L   #count,d0
00002006                          6          END      $2000
```

- **NOTES**
  - The value of $100 replaces the symbol in the binary code
  - The use of EQU directives is encouraged because
    - makes program more readable
    - makes programs easier to maintain

The END directive

- **FUNCTION**
  - Used at the end of the source program
  - Statements following the END directive are not processed by the assembler
- **EXAMPLE**

```
00001000                        1     ORG      $1000
00001000  203C 00000012         2     MOVE.L   #$12,d0
00001006                        3     END      $1000
```

- **NOTES**
  - The label of the END directive (optional) represents the entry point for the program
  - The address of the entry point is used by debuggers, loaders, conversion utilities, and so on, to identify the starting address of the program

The DC directive

- **FUNCTION (DEFINE CONSTANT)**
  - Places data constants WITHIN A PROGRAM
- **EXAMPLE**

```
0000000D                          1 cr        EQU     $0D
00001000                          2           ORG     $1000
00001000  0005FFFF                3 num       DC      5,-1
00001004  05FF                    4 more      DC.B    5,-1
00001006  777269676874            5 name      DC.B    'wright'
0000100C  0D00                    6 var       DC.B    cr,0
0000100E                          7           END     $1000
```

- **NOTES**
  - For words and longwords, the assembler adjusts the address of the constant to ensure proper alignment.
  - ASCII characters defined as words are left-justified within the word

The DS directive

- **FUNCTION (DEFINE STORAGE)**
  - Reserves RAM storage for use during execution of the program.
- **EXAMPLE**

```
00000004                  1 length     EQU     4
00001000                  2            ORG     $1000
00001000                  3 buffer     DS.B    length
00001004  FF              4 temp       DC.B    $FF
00001005                  5            END
```

- **NOTES**
  - The memory locations reserved for **`buffer`** are not initialized, they **will** contain garbage data

The EXTERN/PUBLIC directives

- **FUNCTION**
  - Used when a program is split over multiple files (modules)
- **EXAMPLE**

MAIN.ASM

```
        EXTERN   SQRT
        MOVE.W   #100,D7
        BSR      SQRT
        …
        …
        …
```

SUBS.ASM

```
        PUBLIC   SQRT
SQRT    ;subroutine is
        ;defined here
        …
        …
        ...
        RTS
```

- **NOTES**
  - MAIN.ASM contains the code of a main program, whereas SUBS.ASM contains the subroutines, which will typically be shared among several main programs.

Instruction categories

- **Data movement**
  - Move operands (data) among memory locations or registers
- **Integer arithmetic**
  - Addition, subtraction, multiply, divide, ...
- **Boolean**
  - AND, OR, XOR, NOT, ...
- **Shift and rotate**
  - Arithmetic-shift, logical-shift, rotate
- **Bit manipulation**
  - Bit test, bit set, bit clear, ...
- **Binary Coded Decimal**
  - Add, subtract and negate in BCD notation
- **Program flow**
  - Branch, jump and return
- **System control**
  - Miscellaneous: trap, reset, SR/CCR manipulation, ...

Data movement

INSTR.	DESCRIPTION	EXAMPLE	

MOVE	Copies an 8-, 16- or 32-bit value from one memory location or register to another memory location or register	`MOVE.B #$8C,D0`
		`MOVE.W #$8C,D0`
		`MOVE.L #$8C,D0`
			[D0]←$XXXXXX8C
			[D0]←$XXXX008C
			[D0]←$0000008C

MOVEA	Copies a source operand to an **address register.** MOVEA operates only on words or longwords. MOVEA.W sign-extends the 16-bit operand to 32 bits.	`MOVEA.W #$8C00,A0` `MOVEA.L #$8C00,AO`	[A0]←$FFFF8C00 [A0]←$00008C00
MOVEQ	Copies a 8-bit signed value in the range –128 to +127 to one of the eight data registers. The data to be moved is sign-extended before it is copied to its destination	`MOVEQ #-3,D0` `MOVEQ #4,D0`	[D0]←$FFFFFFFD [D0]←$00000004
MOVEM	Transfers the contents of a **group** of registers specified by a list. The list of registers is defined as A_i-A_j/D_p-D_q. MOVEM operates only on words or longwords.	`MOVEM.L A0-A3/D0-D7,-(A7)`	;copies all working ;registers to stack

Integer arithmetic

INSTR.	DESCRIPTION	EXAMPLE	

ADD×
SUB×
	ADD×/SUB× add/subtract the contents of a source to/from the contents of a destination and deposits the result in the destination location. Direct memory-to-memory operations are not permitted. Assume `[D0]=$11118000` and `[D1]=$11110123`.	`ADD.W D0,D1`
		`ADD.L D0,D1`
		`ADDQ #N,D1`
		`SUB.L D1,D0`
			`;[D1]←$11118123`
			`;[D1]←$22228123`
			`;N∈[1,8]`
			`;[D0]←$00007EDD`

MULU **MULS**	MULU (multiply unsigned) forms the product of two 16-bit integers. The 32-bit destination must be a data register. MULS is similar but treats data as signed. Assume [D0]=$ABCD8000.	`MULU #$0800,D0`	`;[D0]←$00400000`
DIVU **DIVS**	DIVU (divide unsigned) works with a 32-bit dividend and a 16-bit divisor. The dividend must be a data register. The 16-bit result is stored in the low word of the destination, and the 16-bit remainder in the high word. DIVS is similar but treats data as signed. Assume [D0]=$0000000E, 14_{10}.	`DIVS #-3,D0`	`;[D0]←$0002FFFC`

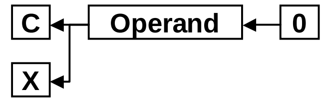
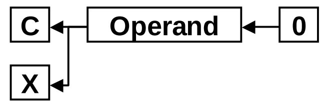
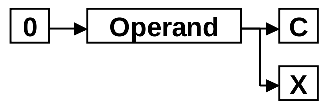
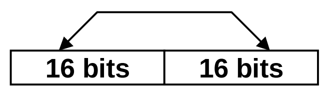
CLR
NEG
	CRL (clear) writes zeros into the destination operand. **NEG** (negate) performs a 2s complement operation on the destination data--subtracts it from zero. Assume [D0]=$1234B021.	`CLR.B D0`
		`CLR.L D0`
		`NEG.W D0`
			`;[D0]←$1234B000`
			`;[D0]←$00000000`
			`;[D0]←$12344FDF`

EXT	Sign-extend increases the bit-size of a signed integer. **EXT.W** converts an 8-bit into a 16-bit, and **EXT.L** converts a 16-bit into a 32-bit. Assume [D0]=$1234B021.	`EXT.W D0` `EXT.L D0`	`;[D0]←$12340021` `;[D0]←$FFFFB021`

Boolean

INSTR.	DESCRIPTION	EXAMPLE	
AND **ANDI**	Bit-wise logical AND operation. Normally used to **clear**, or **mask**, certain bits in a destination operand.	`ANDI.B #%0111111,D0`	`;clear the 8th least` `;significant bit of D0`
OR **ORI**	Bit-wise logical OR operation. Normally used to **set** certain bits in a destination operand.	`ORI.B #%10101010,D0`	`;set even bits of D0` `;lowest byte`
EOR **EORI**	Bit-wise logical XOR operation.	`EOR.B #%11111111,D0`	`;XOR of the lowest byte of D0`
NOT	Bit-wise NOT operation. Assume `[D0]=$1234F0F0.`	`NOT.W D0`	`;[D0]←$12340F0F`
TST	Similar to `CMP #0, operand`	`TST D0`	`;update N,Z and clear V,C`

Shift and rotate

INSTR.	OPERATION	BIT MOVEMENT
ASL	Arithmetic shift left	
ASR	Arithmetic shift right	
LSL	Logic shift left	
LSR	Logic shift right	
ROL	Rotate left	
ROR	Rotate right	
SWAP	Swap words of a longword	

Bit manipulation

INSTR.	DESCRIPTION	EXAMPLE (Assume `[D0]=$00000009`)	
BSET	**Bit test and set** Causes the Z-bit to be set if the specified bit is zero and then forces the specified bit of the operand to be set to one	`BSET #2, D0`	`;[D0]←$0000000D and [Z]←1`
BCLR	**Bit test and clear** works like BSET except that the specified bit is cleared (forced to zero) after it has been tested	`BCLR #0, D0`	`;[D0]←$00000008 and [Z]←0`
BCHG	**Bit test and change** causes the value of the specified bit to be reflected in the Z-bit and then toggles (inverts) the state of the specified bit	`BCHG #4, D0`	`;[D0]←$00000019 and [Z]←1`
BTST	**Bit test** reflects the value of the specified bit in the Z-bit	`BTST #2, D0`	`;[Z]←1`

Binary Coded Decimal

INSTR.	DESCRIPTION	EXAMPLE (Assume `[X]=0, [D0]=48, [D1]=21`)	
ABCD	Adds the source operand and the X-bit to the destination operand using BCD arithmetic. This is a BYTE operation only; the X-bit is used to provide a mechanism for multi-byte BCD operations.	`ABCD D0,D1`	`;[D1]←00000069`
SBCD	Subtract the source operand and the X-bit from the destination operand using BCD arithmetic. This is a BYTE operation only, so the X-bit is used to provide a mechanism for multi-byte BCD operations.	`SBCD D1,D0`	`;[D0]←00000027`
NBCD	Subtract the destination operand and the X-bit from zero.	`NBCD D1`	`;[D1]←00000052` `;[X]←1, [V]←1, [C]←1`

Program flow (details in Lecture 5)

INSTR.	DESCRIPTION
BRA	BRA (branch always) implements an unconditional branch, relative to the PC. The offset is expressed as an 8- or 16-bit signed integer. If the destination is outside of a 16-bit signed integer, BRA **cannot** be used.
B*cc*	B*cc* (branch conditional) is used whenever program execution must follow one of two paths depending on a condition. The condition is specified by the mnemonic *cc*. The offset is expressed as an 8- or 16-bit signed integer. If the destination is outside of a 16-bit signed integer, B*cc* **cannot** be used.
BSR **RTS**	BSR branches to a subroutine. The PC is saved on the stack before loading the PC with the new value. RTS is use to return from the subroutine by restoring the PC from the stack.
JMP	JMP (jump) is similar to BRA. The only difference is that BRA uses only relative addressing, whereas JMP has more addressing modes, including absolute address (see reference manual).

cc	CONDITION	BRANCH TAKEN IF
CC	Carry clear	C=0
CS	Carry set	C=1
NE	Not equal	Z=0
EQ	Equal	Z=1
PL	Plus	N=0
MI	Minus	N=1
HI	Higher than	$\overline{C}\overline{Z} = 1$
LS	Lower than or same as	C+Z=1
GT	Greater than	$NV\overline{Z} + \overline{N}\overline{V}\overline{Z} = 1$
LT	Less than	$N\overline{V} + \overline{N}V = 1$
GE	Greater than or equal to	$N\overline{V} + \overline{N}V = 0$
LE	Less than or equal to	$Z + (N\overline{V} + \overline{N}V) = 1$
VC	Overflow clear	V=0
VS	Overflow set	V=1
T	Always true	Always
F	Always false	Never

System control

INSTR.	DESCRIPTION
MOVE **ANDI** **ORI** **EORI**	Unique variations of MOVE, AND, OR and EOR that allow altering the bits in the status and condition code registers.
TRAP	TRAP performs three operations: (1) pushes the PC and SR to the stack, (2) sets the execution mode to supervisor and (3) loads the PC with a new value read from a vector table
STOP **RESET**	STOP loads the SR with an immediate operand and stops the CPU. RESET asserts the CPU's $\overline{\text{RESET}}$ line for 124 cycles. If STOP or RESET are executed in user mode, a *privilege violation* occurs.