Addressing Modes

- **The Effective Address:**

When an operand is in memory, the instruction must somehow specify the address.

The actual address of the operand in memory is called the *effective address*. (16 bits in HCS12)

- **Addressing mode:**

Means how the HC12 calculates the effective address.

Addressing Modes (Six)

1. **Inherent addressing mode (INH)**
2. **Immediate addressing mode (IMM)**
3. **Direct page addressing mode (DIR)**
4. **Extended addressing mode (EXT)**
5. **PC relative addressing mode (REL)**
6. **Indexed addressing mode (IDX)**

Addressing Modes: *Inherent*

- No extra bytes to specify operands
  - The instructions either do not need operands
  - or all operands are CPU registers.
- Operands are implied by the opcode.

Examples

• NOP	; no operand needed
• INX	; Operand is a CPU register (index reg. X)
• DECA	; Operand is a CPU register (acc. A)

Addressing Modes: *Immediate*

- Operand is part of the instruction.
- Used when an operand is a constant known at the time the program is written.

Examples

```
• LDAA    #$55       ; one-byte operand
• LDX     #$1000     ; 16-bit operand
• LDY     #$67       ; 16-bit operand ($0067)
```

Addressing Modes: *Direct*

Can only specify memory locations in the range 0 - 255.

(zero-page addressing)

Uses only one byte to specify the operand address.

(saves memory & execution time)

Examples

- LDAA $20
- LDX $40 ;16-bit word is read from addresses $0040 and $0041

Addressing Modes: *Extended*

The full 16-bit address is provided in the instruction.

Can access any location in the 64-Kbyte memory.

Examples:

- LDAA $4000
- LDX $FE60 ;16-bit word is read from addresses $FE60 and $FE61

Addressing Modes: *Relative* (1 of 3)

Used only by branch instructions (conditional & unconditional)

Conditional branch instructions (short and long) use exclusively relative mode:

Short branch :

8-bit opcode	8-bit signed offset (-128 to +127)

Long branch :

8-bit prebyte	8-bit opcode	16-bit signed offset (-32,768 to +32,767)

- Loop primitive instructions [DBEQ, DBNE, IBEQ, IBNE, TBEQ, TBNE] use 9-bit offset (-256 to +255)

Note: The programmer uses a symbol to specify the branch target and the assembler will figure out the actual branch offset (distance) from the instruction that follows branch instruction.

Conditional branch instructions test bits in the **CCR**:

- **If** the bits are in a specified state, the offset is added to contents of the PC to form an effective address to branch to.
- **else**, execution continues with the next instruction.

Addressing Modes: *Relative* (2 of 3)

Examples:

1- Conditional branches: (test bits in the **CCR**)

```
Loop1:   .
         .
        beq    Loop1    ; Branch if equal (Z = 1)          (one flag is tested)
Loop2:   .
         .
        bhi    Loop2    ; Branch if higher (if C + Z = 0)  (two flags are tested)
                                                           (Used with unsigned numbers)
```

Using 8-bit or 16-bit offset for **Loop1/Loop2** depends on how far the instruction **beq/bhi** is from instruction labeled **Loop1:/Loop2:** … a decision made by the assembler.

2- Unconditional branches:

```
Loop1:   .
         .
         bra   Loop1    ; Branch always
Loop2:   bra   Loop2    ; Infinite loop    (wait for an interrupt or RESET)
```

Question: What is the value of the offset for loop2 for infinite loop?
What should be that value if long branch instruction LBRA is used?

Addressing Modes: *Relative* (3 of 3)

Bit-condition branch instructions test *some* bits in a **memory byte**:

- **If** each bit in memory that corresponds to a 1 in the mask is either set (in BRSET) or clear (in BRCLR), the 8-bit offset is added to contents of the PC to form an effective address to branch to.
- **else,** execution continues with the next instruction.

The **memory byte** is accessed using various addressing modes.

An 8-bit mask operand is used to test the bits.

Relative mode (short) is used to specify the branch target.

Example:

- **BRSET FOO , #$03 , THERE**

```
                    ; FOO     can be direct, extended, or indexed
                    ; #$03    immediate
                    ; THERE   relative short
```

Therefore BRSET and BRCLR instructions can be 4, 5, or 6 bytes long.

Addressing Modes: *Indexed* (1 of 4)

This mode uses the sum of a base index register and an offset to specify the address of an operand.

The base index register can be X, Y, PC, or SP

The offset can be:

- a 5-bit, 9-bit, and 16-bit signed value specified by the instruction or
- the value in accumulator A, B, or D.

Indirect indexing with 16-bit offset.
The 16-bit offset is added to the base index register to form the address of a memory location that contains a pointer to the operand in memory.
The offset can be:

- a signed value specified by the instruction or
- the value in accumulator D

Automatic pre/post increment/decrement of the base index register by −8 to +8 are options (except with PC).

Addressing Modes: *Indexed* (2 of 4)

Examples: Constant offset

- 5-bit: The range of the offset is from -16 to +15.

```
stab  –8,Y
stab   0,Y
```

- 9-bit: The range of the offset is from -256 to +255.

```
ldab  –200,X
```

- 16-bit: This mode allows access to the 64-KB range.

```
staa  4000,SP
```

Examples: Accumulator offset

```
ldaa  B,X
sts   D,X
```

Examples: Indirect Indexed Addressing

– The square brackets distinguish this addressing mode from the 16-bit constant offset indexing.

```
ldaa  [10,X]
jmp   [D,PC]
```

Addressing Modes: *Indexed* (3 of 4)

Auto pre/post increment/increment Indexed Addressing

- The base index register can be X, Y, or SP. (Cannot use PC)
- The index register can be incremented or decremented by an integer value either pre (**before**) or post (**after**) indexing taking place.
- The index register retains the changed value after indexing.
- The value to be incremented or decremented is in the ranges -8 thru -1 or 1 thru 8.
- The value needs to be related to the size of the operand or the current instruction.

Examples: (assume that Y=2345)

```
staa  1,Y+          ; Store A at 2345, then Y=2346  (post increment)
staa  4,Y –         ; Store A at 2345, then Y=2341
staa  4,+Y          ; Y=2349, then store A at 2349
staa  1,– Y         ; Y=2344, then store A at 2344  (pre decrement)
```

- **Note:** **ldaa [2,X] does not change the value in X,**
  while ldaa 2,+X changes it by 2.

Addressing Modes: *Indexed* (4 of 4)

Example: Computed GOTO

```
        JMP     [D,PC]
GO1     DC.W    PLACE1
GO2     DC.W    PLACE2
GO3     DC.W    PLACE3
```

The values PLACE1, PLACE2, PLACE2 are addresses of potential destinations of the JMP At the time this instruction is executed, PC has the address corresponding to GO1, and D holds one of the values $0000, $0002, or $0004 (determined by the program some time before the JMP).

Assume that the value in D is $0002. The JMP instruction adds the values in D and PC to form the address of GO2. Next the CPU reads the address PLACE2 from memory at GO2 and jumps to PLACE2.

The locations of PLACE1 through PLACE3 were known at the time of program assembly but the destination of the JMP depends upon the value in D computed during program execution.

Data Types (1 of 2)

68HCS12 supports:

- Bits
- 5-bit signed integers
- 8-bit signed and unsigned integers
- 9-bit signed integers
- 8-bit, 2-digit BCD numbers
- 16-bit signed and unsigned integers
- 16-bit effective address
- 32-bit signed and unsigned integers

A multi-byte integer is stored in memory from most significant to least significant bytes starting from low to higher address.

A number can be represented in binary, octal, decimal, or hex format according to the prefix:

– Binary	%11001010
– Octal	@4725
– Decimal	no prefix
– Hexadecimal	$

Data Types (2 of 2)

Negative integers are represented in two’s complement form.

Five-bit and 9-bit signed integers are used only as offsets for indexed addressing modes.

Sixteen-bit effective addresses are formed during addressing mode computations.

Thirty-two-bit integer dividends are used by extended division instructions.

Extended multiply and extended multiply-and-accumulate instructions produce 32-bit products.