ECE331 Handout 6- Advanced ASM

This handout describes 68HC12 Instructions and Address Modes that will NOT becoved in class or on exams, but you can use them in programming class assignments if you want to learn them on your own.

Advanced Address Modes

Indexed-Indirect

- offset from *reference* stored in accumulator
- offset + reference points to address containing data (not to data itself)
- defined by putting operands in brackets [ ]

Example:

LDAA [D,X] ; {A ← <D+IX>}

accA loaded with value at address specified by D+IX

Indexed-Immediate with Increment

- adjusts *reference* by *offset*
- reference can be IX, IY, SP (not PC)
- adjustment can be before or after instruction execution
  - o pre-increment/decrement (+)
  - o post-increment/decrement (-)
- defined by putting +/- in operand
  - o before operand = pre-decrement
  - o afer operand = post-decrement

Examples:

pre-increment

LDAB $3,+Y

1) {IY ← IY + $3}

2) accB loads value from address specified by <new IY>

post-decrement

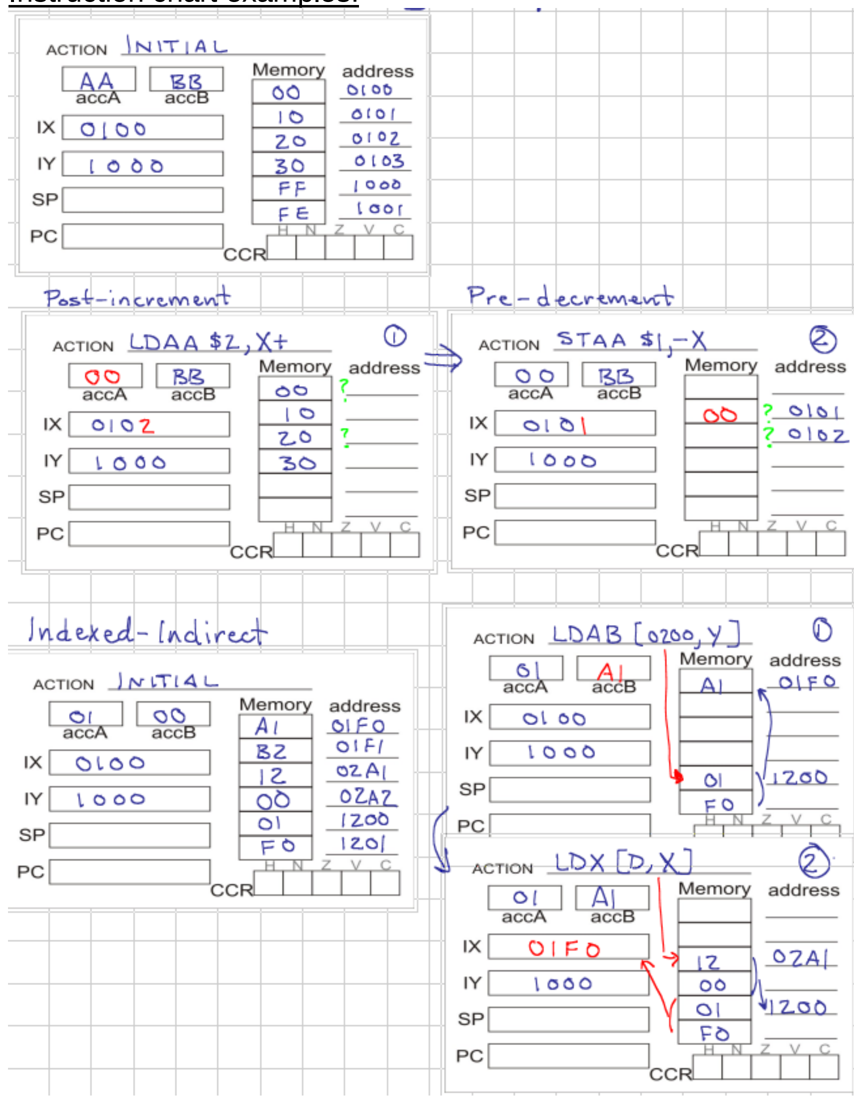
LDAB $2,Y-

1) accB loads value from address specified by <IY>

2) {IY ← IY - $2}

Note in pre-increment, data is loaded after IY is adjusted while in post-increment data is loaded before IY is adjusted.

Instruction chart examples:



Advanced Branch Instructions

Loop Primitive Instructions

- simultaneously branch and increment counter
- counter can be in A, B, D, X, Y, SP
- can increment, decrement, or test
- can branch if counter = 0 or ≠ 0

Mnemonic	Function	Equation or Operation
DBEQ cntr, rel	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) - 1 If (counter) = 0, then branch else continue to next instruction
DBNE cntr, rel	Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) - 1 If (counter) ≠ 0, then branch else continue to next instruction
IBEQ cntr, rel	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) + 1 If (counter) = 0, then branch else continue to next instruction
IBNE cntr, rel	Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	counter ← (counter) + 1 If (counter) ≠ 0, then branch else continue to next instruction
TBEQ cntr, rel	Test counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	If (counter) = 0, then branch else continue to next instruction
TBNE cntr, rel	Test counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	If (counter) ≠ 0, then branch else continue to next instruction

Note. 1. **cntr** is the loop counter and can be accumulator A, B, or D and register X, Y, or SP.
2. **rel** is the relative branch offset and is usually a label

Table 2.5 ■ Summary of loop primitive instructions

Example 1

Write a program to add an array of N 8-bit numbers and store the sum at memory location \$800–\$801. Use the ***For** i* **–** *n1* ***to** n2* ***do*** looping construct.

Solution: We will use variable **i** as the array index. This variable can also be used to keep track of the number of iterations remained to be performed. We will use a two-byte variable **sum** to hold the sum of array elements. The logic flow of the program is illustrated in Figure 2.9.

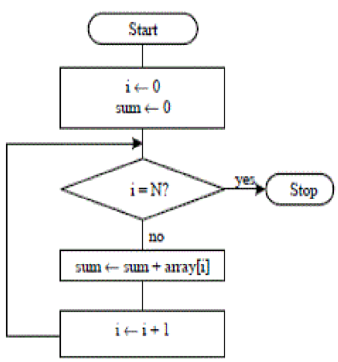


Figure 2.9 ■ Logic flow of example 2.14

The program is a direct translation of the flowchart shown in Figure 2.9.

```
N       equ     20                      ; array count
        org     $800                    ; starting address of on-chip SRAM
sum     rmb     2                       ; array sum
i       rmb     1                       ; array index
        org     $1000                   ; starting address of the program
        ldaa    #0
        staa    i                       ; initialize loop (array) index to 0
        staa    sum                     ; initialize sum to 0
        staa    sum+1                   ;    "

loop    ldab    i
        cmpb            #N              ; is i = N?
        beq     done                    ; if done, then branch
        ldx     #array                  ; use index register X as a pointer to the array
        abx                             ; compute the address of array[i]
        ldab    0,x                     ; place array[i] in B
        ldy     sum                     ; place sum in Y
        aby                             ; compute sum <- sum + array[i]
        sty     sum                     ; update sum
        inc     i                       ; increment the loop count by 1
        bra     loop
done    swi                             ; return to D-Bug12 monitor
; the array is defined in the following statement
array   db      1,2,3,4,5,6,7 , 8,9,10,11,12,13,14,15,16,17,18,19,20
        end
```

It is a common mistake for an assembly language programmer to forget to update the variable in memory. For example, we will not get the correct value for **sum** if we did not add the instruction **sty sum** in the program shown in Example 2.14.

Loop primitive instructions are especially suitable for implementing the **repeat** S **until** C looping construct as demonstrated in the following example.

Example 2

Write a program to find the maximum element from an array of N 8-bit elements using the ***repeat S until C*** looping construct.

Solution: We will use the variable **i** as the array index and also as the loop count. The variable **max_val** will be used to hold the array maximum. The logic flow of the program is shown in Figure 2.10.

The program is as follows:

```
N       equ     20              ; array count
        org     $800            ; starting address of on-chip SRAM
max_val rmb     1               ; memory location to hold array max
        org     $1000           ; starting address of program
        ldaa    array           ; set array[0] as the temporary array max
        staa    max_val         ;    "
        ldx     #array+N-1      ; start from the end of the array
        ldab    #N-1            ; use B to hold variable i and initialize it to N-1
loop    ldaa    max_val
        cmpa    0,x             ; compare max_val with array[i]
        bge     chk_end         ; no update if max_val is larger
        ldaa    0,x             ; update max_val
        staa    max_val         ;    "
chk_end dex                     ; move the array pointer
        dbne    b,loop          ; decrement the loop count, branch if not zero yet.
forever bra     forever
array   db      1,3,5,6,19,41,53,28,13,42,76,14,20,54,64,74,29,33,41,45
        end
```