Indexed Addressing
Addressing Modes

- Indexed addressing mode
  - Sometimes you want the effective address to be determined at run time (when the program executes), not at assembly time.
  - In this mode, the index registers (X or Y) are used to form the address of the operand.
  - Example (constant offset)

```
LDAA 3, X ; A <- [3 + [X]]
```

*load A with contents of memory at the address equal to the sum of 3 and X*

Assume X contains

| X   | $0820 |

**Effective address is $0823**

<table>
<thead>
<tr>
<th>$0820</th>
<th>$0821</th>
<th>$0822</th>
<th>$0823</th>
</tr>
</thead>
</table>

A is loaded with contents of memory at $0823

| A    | $1D   |

$1D$0820$0821$0822$0823$1D
Tables and Indexed Addressing

- Indexed addressing mode is very useful for reading and writing to “tables”; i.e., blocks of memory
- Example:
  - Register X contains the starting address of a table of data. The table is 3 bytes long. We want to copy the table to the table whose starting address is given in register Y.

Could do it this way:

LDAA 0, X
STAA 0, Y
LDAA 1, X
STAA 1, Y
LDAA 2, X
STAA 2, Y

Or, even simpler:

MOV B 0, X, 0, Y
MOV B 1, X, 1, Y
MOV B 2, X, 2, Y
Example – Clear a Table

• Problem statement
  – Clear (i.e., set to zero) memory locations in a table
  – The table starts at $3800 and is 255 bytes long

• Approach
  – Load an index register (say, X) with the starting address of the table
  – Store a zero to the address contained in X
  – Increment X, and repeat 255 times

• Assembly code:

```
ldx  #$3800         ; starting address of table
ldaa  #255         ; use accumulator A as a loop counter
loop
  movb  #0, 0,x    ; clear next byte in memory
  inx              ; increment x
  deca             ; decrement loop counter
bne   loop         ; keep looping as long as loop counter not = 0
```
Example (continued)

• Another way to do this:
  – Recall that the “dbne” instruction combines a decrement instruction and a “bne” instruction

```assembly
ldx  #$3800 ; starting address of table
ldaa #255  ; use accumulator A as a loop counter
loop movb #0, 0,x ; clear next byte in memory
   inx       ; increment x
   dbne a,loop ; decrement A and loop back if A not = 0
```
Example – Copy a Table

• Problem statement
  – Copy a table from one location ($1000) to another ($2000)
  – The table is 500 bytes long

• Approach
  – Load index register X with the starting address of the first table
  – Load index register Y with the starting address of the second table
  – Load D with the count of 500
  – Move a byte from the address pointed to by (X) to the address pointed to by (Y)
  – Decrement D, and repeat until D = 0

• Assembly code:

```
ldx #$1000      ; starting address of Table1
ldy #$2000      ; starting address of Table2
ldd #500        ; loop counter
loop movb 0,x,0,y ; copy byte from Table1 to Table2
    inx ; increment X
    iny ; increment Y
    dbne d,loop ; keep looping while D not = 0
```
Example (continued)

• What if we wanted to copy 500 words instead of 500 bytes? A “word” is 2 bytes.

• You could just change the preceding program to copy 1000 bytes.

• Or, change the program to copy 500 words:

```
ldx #$1000 ; starting address of Table1
ldy #$2000 ; starting address of Table2
ldd #$500 ; loop counter

loop movw 0,x,0,y ; copy word from Table1 to Table2
  inx ; increment X twice
  inx
  iny ; increment Y twice
  iny
  dbne d,loop ; keep looping while D not = 0
```
Example – Gray Code counter

- **Problem statement:**
  - Continuously count up (delay a short time after each increment) from 0 through 7, and then wrap around back to 0
  - Display the count as a 3 bit Gray code on Port T
  - Recall a 3 bit Gray code is: 000, 001, 011, 010, 110, 111, 101, 100

- **Approach:**
  - Make a table and initialize it with the Gray codes
  - Have an infinite loop that increments a counter
  - Get the value from the table corresponding to the count, and write it to PortT
  - When the counter reaches 8, reset it back to zero
Algorithm pseudocode

Initialize Table with Gray codes (note - this can be done by the assembler at load time, and the table can be stored in ROM)

Set up Port T, bits 0..2, for output
Initialize the counter n=0
Repeat forever
  Get the nth value from the table; ie Table(n)
  Write that value to PortT
  Delay a short time
  increment n
  if n==8
    n = 0
  end
end
Assembly

movb #$07, DDRT ; Set up PT0..PT2 for output
ldaa #8 ; Use A as the counter
ldx #GrayTable ; Get starting address of table

loop movb 0,x,PTT ; Get next byte from table, write to Port T
; (optionally insert code to delay awhile)
inx ; Point to next byte in table
dbne a,loop ; if A not equal 0, go back to loop

ldaa #8 ; Reset loop counter
ldx #GrayTable ; Reset starting address of table
bra loop

GrayTable
dc.b %000,%001,%011,%010,%110,%111,%101,%100

Note – the table is loaded into ROM immediately after the machine code
Summary / Questions

• Each type of machine code instruction (such as “load A”) can have up to 6 types of addressing modes - indexed addressing is one of these modes

• How do you know what types of modes each instruction can support?

• What is the advantage of indexed addressing? Could you get by without it?