14 – ARRAYS & INDIRECT ADDRESSING

COMP256 – COMPUTING ABSTRACTIONS

DICKINSON COLLEGE

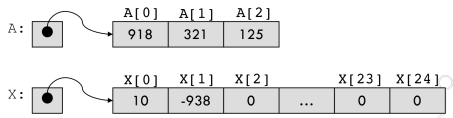
HLL ARRAY ABSTRACTIONS

- Arrays are an abstraction created by HLLs that allow us to store and process ordered lists of information.

HLL Array Code

```
int A[] = {918, 321, 125};
int X[] = new int[25];
…
x[0] = 10;
X[1] = -938;
int z = A[2];
```

Array Abstraction



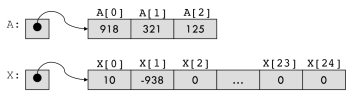
ARRAY'S AND REFERENCES IN MAIN MEMORY

- Like all program data, arrays and references are stored in the computer's main memory.

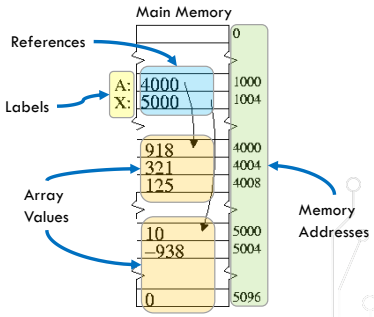
HLL Array Code

```
int A[] = {918, 321, 125};
int X[] = new int[25];
…
x[0] = 10;
X[1] = -938;
int z = A[2];
```

Array Abstraction



Physical Array Implementation



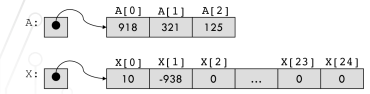
ALLOCATING ARRAYS & REFERENCES

- Arrays are allocated in assembly language using the `.word` or `.space` assembler type directives.

HLL Array Allocation

```
int A[] = {918, 321, 125};
int X[] = new int[25];
…
```

Array Abstraction



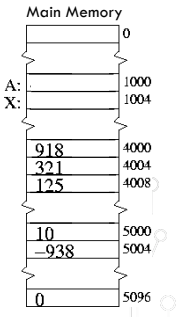
Assembly Array Allocation

```
A:     .word
X:     .word
AARR:  .word 918 321 125
XARR:  .space 100
       …
```

Allocate 100 bytes of space.

Allocate and initialize three 32- bit words

Physical Array Implementation



THE SYMBOL TABLE

- The assembler uses a data structure called a **symbol table** that tracks the addresses corresponding to the labels in the program.

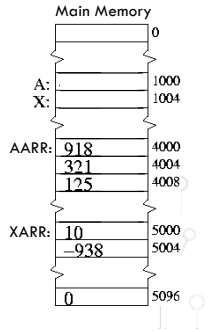
Assembly Array Allocation

```
A:     .word
X:     .word
AARR:  .word 918 321 125
XARR:  .space 100
       …
```

Symbol Table

Symbol	Address
A	1000
X	1004
AARR	4000
XARR	5000

Physical Array Implementation



CREATING REFERENCES

- References are created using **immediate label addressing mode** instructions, which load the value of a label (i.e. a memory address) *immediately* into a register.

Assembly Array Allocation

```
A:     .word
X:     .word
AARR:  .word 918 321 125
XARR:  .space 100
       …
       LOAD R0 #AARR
       STORE R0 A
       LOAD R1 #XARR
       STORE R1 X
```

Immediate Label Addressing Mode Instructions

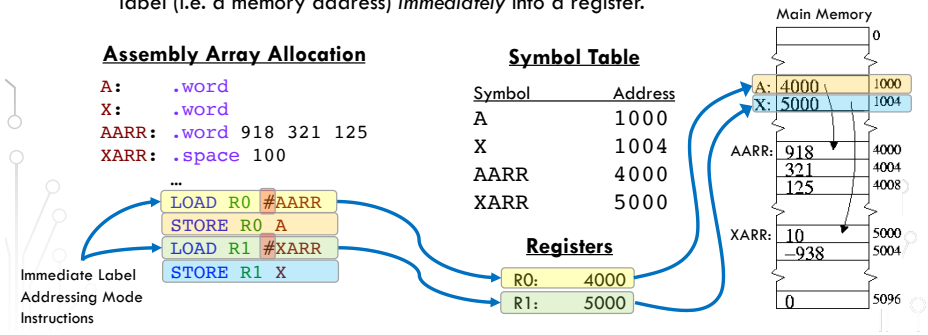
Symbol Table

Symbol	Address
A	1000
X	1004
AARR	4000
XARR	5000

Registers

R0:	4000
R1:	5000

Physical Array Implementation



ACCESSING ARRAY ELEMENTS

- Array elements are accessed using **register indirect addressing mode** instructions, which use the value in a register *indirectly* as a memory address.

Assembly Array Accesses

```
A:     .word
AARR:  .word 918 321 125
       …
       LOAD R0 #AARR
       STORE R0 A
```

Register Indirect Addressing Mode Instructions

```
LOAD R1 R0      * R1←A[0]
                * R1←MM[R0]
LOAD R2 R0 +8   * R2←A[2]
                * R2←MM[R0+8]
```

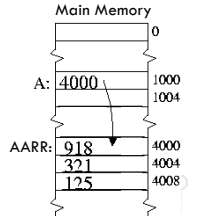
Symbol Table

Symbol	Address
A	1000
AARR	4000

Registers

R0:	4000
R1:	918
R2:	125

Physical Array Implementation



LOAD AND STORE ADDRESSING MODES

- The **addressing mode** (immediate, direct, register indirect) of an instruction indicates how the operands are to be used to locate the data.

Instruction	Example	Meaning	Addressing Mode
LOAD R #	LOAD R1 #27	R1 ← 27	Immediate
LOAD R L	LOAD R1 X	R1 ← MM[X]	Direct
LOAD R #L	LOAD R1 #XARR	R1 ← XARR	Immediate Label
LOAD R R	LOAD R1 R0	R1 ← MM[R0]	Register Indirect
LOAD R R +	LOAD R1 R0 +4	R1 ← MM[R0+4]	Register Indirect
STORE R L	STORE R1 X	MM[X] ← R1	Direct Mode
STORE R R	STORE R1 R0	MM[R0] ← R1	Register Indirect
STORE R R +	STORE R1 R0 +8	MM[R0+8] ← R1	Register Indirect

ACTIVITY

- Give the assembly translation of the missing HLL statements.

```
// HLL Code
int v[] = new int[100]

int p;



p = v[0];

v[5]=10;

v[10]=p;

v[11]=v[12]
```

```
* Assembly Code
VARR: .space 400
V:    .word
P:    .word

      LOAD R0 #VARR
      STORE R0 V
```

ACTIVITY

- Give the assembly translation of the missing HLL statements.

```
// HLL Code
int v[] = new int[100]

int p;



p = v[0];

v[5]=10;

v[10]=p;

v[11]=v[12]
```

```
* Assembly Code
VARR: .space 400
V:    .word
P:    .word

      LOAD R0 #VARR
      STORE R0 V

      LOAD R1 R0
      STORE R1 P

      LOAD R1 #10
      STORE R1 R0 +20

      LOAD R1 P
      STORE R1 R0 +40

      LOAD R1 R0 +48
      STORE R1 R0 +44
```

PROCESSING AN ARRAY

```
// HLL Code
int X[] = new int[25];



for (int i=0; i<25; i++) {
  X[i] = 2*i;
}
```

```
* Assembly Code
XARR: .space 100        * 100 bytes = 25 32-bit int
X:    .word

      LOAD R0 #XARR     * Setup reference to X
      STORE R0 X

      LOAD R1 #0        * R1 is i
      LOAD R2 #25       * for condition check.
      JUMP COND

TOP:  SHL R3 R1         * R3 = 2*i
      STORE R3 R0       * X[i] = R3

      ADD R0 R0 #4      * Advance to next index in X

      ADD R1 R1 #1      * i++

COND: BLT R1 R2 TOP     * i < 25?

      HALT
```