Assembly Language

CSE 132

Logistics

- Last 3 weeks of semester will focus on assembly language
  - Lecture and studio today and next week
  - Two assignments, first starting this Wed.
  - There will be a quiz on this material
- Last lecture and studio will be review for final
  - Material will be cumulative
  - Tuesday, May 10, 10:30am-12:30pm, Lab Sci 300
- Quiz assigned today, due Wed. evening
  - Tell us if you have a conflict with final exam time

General Form

```
label:  opcode  operands     comment
```

- Label is optional
- Opcode is the specific instruction (e.g., add)
- Operands specify data for operation
  - AVR is 2-operand machine, 1st operand is dest.
- Comments use different notations
  - Many assemblers (incl. AVR) `; comment`
  - Or some other notation, e.g., `# comment`

Pseudo-operations

Pseudo-ops are commands to assembler

.text	means "text section", or instructions are next
.data	means "data section"
.byte	reserves data storage

```
var:  .byte  10
```

reserves one byte, initializes it to 10, and makes var a label that is address of byte

Example

```
byte rshift2(byte x) {
      return(x >> 2);
}
```

```
      .text             ;code segment follows
      .global rshift2   ;tell linker about rshift2
rshift2:
      lsr r24           ;do actual work
      lsr r24           ;result is in r24
      ldi r25, 0        ;return value in r25:r24
      ret               ;return
```

Data Segment

```
byte x;          //x declared as single byte
```

In assembly looks like this:

```
      .data     ;data segment follows
x:    .byte     ;reserve one byte for x
```

As in C, no automatic initialization takes place, this is the programmer's responsibility!

Example (2)

```
byte xRshift2() {//x declared elsewhere
        return(x >> 2);
}

xRshift2:
        ldi r30, lo8(x)     ;load addr of x into
        ldi r31, hi8(x)     ;     index reg Z
        ld r24, Z           ;load x into r24
        lsr r24             ;do actual work
        lsr r24             ;result is in r24
        mov r25, r1         ;r1 is normally zero
        ret                 ;return
```

Cautions

- Assembly has no understanding of data type
  - Programmer must handle multi-byte data
  - No conversions, Load (ld) just copies bits in memory to same bits in register
- Addresses are 16 bits
  - Requires two registers (r31:r30) and two loads
    - lo8(x) gives low byte of x, hi8(x) gives high byte of x
    - Use Load Immediate (ldi), because x is the address

Assembly and C

Each can call the other, but assembly routine must follow rules set by C compiler

- r0 is temporary, alter with impunity
- r1 is zero, if changed in assembly, change back
- r2 to r17 and r28 to r29 are callee save
  - Called routine must save if it wishes to use register
- r18 to r27 and r30 to r31 are caller save
  - Calling routine must save register if value is to be preserved across the call

Parameters and Return Values

- Two-byte return values go in r25:r24
- Parameters go in register pairs
  - First parameter in r25:r24
  - Second param. in r23:r22
  - Third param. in r21:r20
  - Etc.
- One-byte return values and parameters use low byte of two-byte register pairs

Multi-byte Data Manipulation

- Use bits in SREG to save intermediate values
- *C* bit (carry) for addition, e.g,
  r9:r8 ← r9:r8 + var

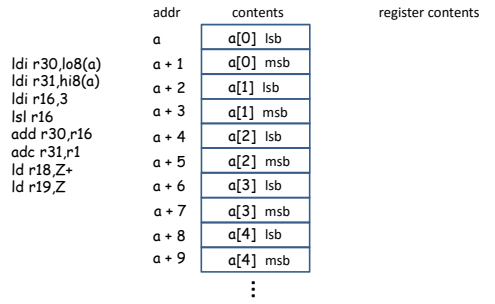
```
lds r4, (var)   ;load var into r5:r4
lds r5, (var+1)
add r8, r4      ;adds lsbits and puts carry in C
adc r9, r5      ;uses carry from prev. add
```

Array indexing

- If the array is declared as follows:
  int a[10];
- And I wish to read a[3] in assembly language
- Use Z (r31:r30) as index register

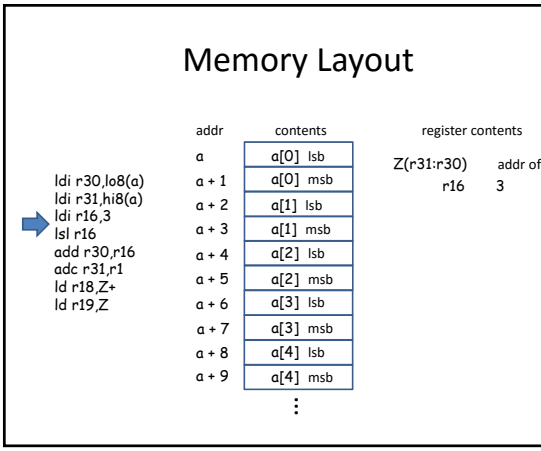
```
ldi r30,lo8(a)    ;use ldi for a pointer, lo8 and hi8 are macros
ldi r31,hi8(a)
ldi r16,3         ;put index value in a register
lsl r16           ;every int takes 2 addresses so multiply index by 2
add r30,r16
adc r31,r1        ;r1 is always zero in compiled C code
ld r18,Z+         ;actually do the load (in two instructions)
ld r19,Z
```

Memory Layout

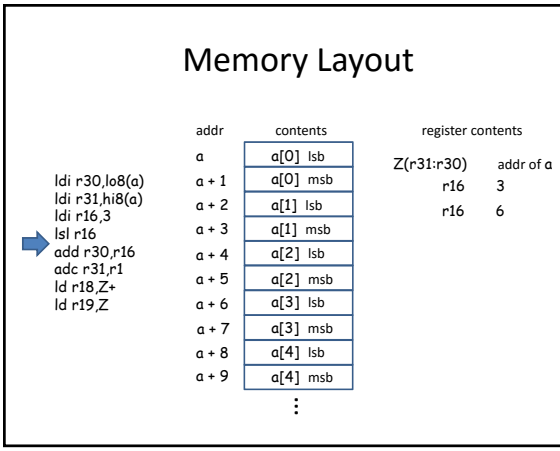


Memory Layout

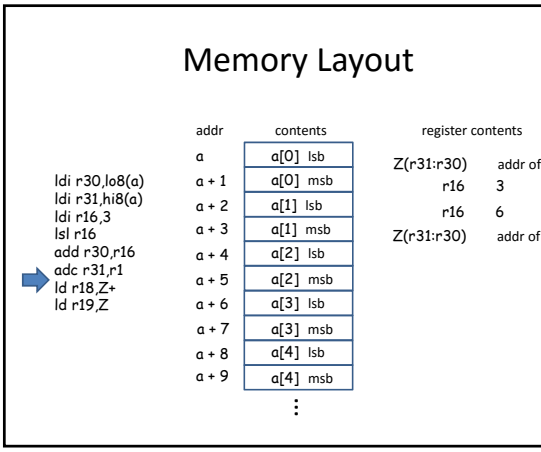
Memory Layout



Memory Layout



Memory Layout



Memory Layout