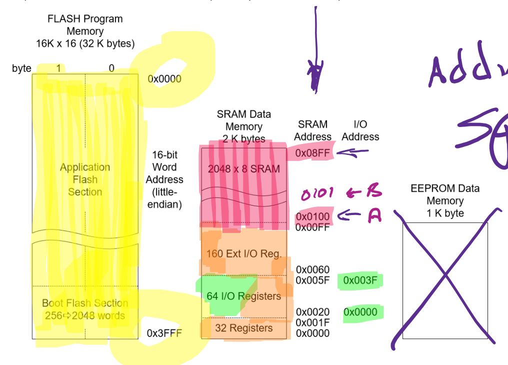
Load-Store Instructions and the ATmega328P Memory Model

- When selecting an addressing mode you should ask yourself where is the operand (data) located within the memory model of the AVR processor and when do I know its address (assembly time or at run time)[1].



Addressing Spaces

Memory Mapped I/O

[1] http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf 8-bit AVR Instruction Set

Load-Store Instructions and Addressing Modes

- When loading and storing data we have several ways to "address" the data.
- The AVR microcontroller supports addressing modes for access to the Program memory (Flash) and Data memory (SRAM, Register file, I/O Memory, and Extended I/O Memory).

	Address Space		
Addressing Mode	Flash Program	SRAM Data	I/O
Immediate	ldi		
Direct		lds, sts	in, out
Indirect	lpm, spm (1)	ld, st (2)	
Indirect with Displacement		ldd, std (3)	

Immediate

opcode destination operand source operand

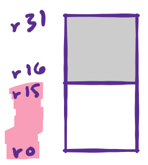
- Data is encoded with the instruction. Operand is therefore located in Flash Program Memory. This is why technically our memory model is a *Modified* Harvard.

$16-12=4$ bits

```
ldi  r16, 0x23      // where ldi = 1110, Rd = 0000₂, and constant K = 00100011₂
```

1110	KKKK	dddd	KKKK
0x E	0010	0000	0011
	2	0	3

- Notice that only four bits (dddd) are set aside for defining destination register Rd. This limits us to $2^4 = 16$ registers. The designers of the AVR processor chose registers 16 to 31 to be these registers (i.e., $16 \le Rd \le 31$).



32 registers
$2^n = 2^5 = 32$

one exception
not orthogonal

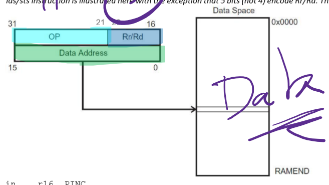
Direct

$2^5 = 32$

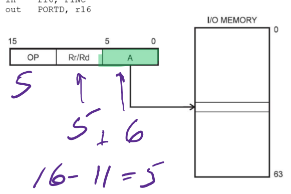
```
lds   r16, A
sts   A, r16
```

Within the AVR family there are two (2) possible lds/sts instructions. A specific family member will have only one lds/sts combination. The ATmega328P lds/sts instruction is illustrated here with the exception that 5 bits (not 4) encode Rr/Rd. This means all 32 registers are available to the lds/sts instruction.

11 5



```
in    r16, PINC
out   PORTD, r16
```



$16 - 11 = 5$

Register-register Instructions

Data Transfer

- Register-register move byte (mov) or word (movw)

Arithmetic and Logic (ALU)

- Two's complement negate (neg), Arithmetic add (add, adc, adiw), subtract (sub, subi, sbc, sbci), and multiply (mul, muls, mulsu, fmul, fmuls, fmulsu)
- Logical not (com), and (and, andi, cbr, tst), or (or, ori, sbr), exclusive or (eor)
- Clear (clr), set (ser), increment (inc), decrement (dec)

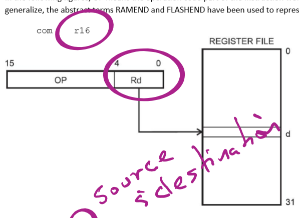
Bit and Bit-Test

- Register logical shift left (lsl) or right(lsr); arithmetic shift right (asr); and rotate left or right (rol, ror)
- Register swap nibble (swap)
- Register bit load (bld) or store (bst) from/to T flag in the Status Register SREG
- I/O Register Clear (cbi) or set (sbi) a bit
- Clear (cl*Flag*) or set (se*Flag*) a Flag bit in the Status Register SREG by name (I, T, H, S, V, N, Z, C) or bit (bclr, bset).

Register Direct

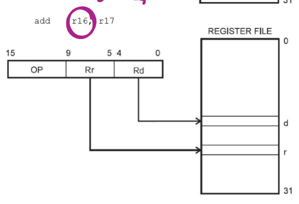
In the following figures, OP means the operation code part of the instruction word. To simplify, not all figures show the exact location of the addressing bits. To generalize, the abstract terms RAMEND and FLASHEND have been used to represent the highest location in data and program space.

```
com   r16
```



source & destination

```
add   r16, r17
```



$r16 = \underline{r16} + r17$

Load-Store Program Example

Write an Assembly program to add two 8-bit numbers.

C = A + B

```
lds   r16, A      ; 1. Load variables
lds   r17, B
add   r16, r17    ; 2 Do something
sts   C, r16      ; 3. Store answer
```

- Identify the operation, source operand, destination operand in the first *Data Transfer* instruction.
- Identify the source/destination operand in the *Arithmetic and Logic* (ALU) instruction.
- What addressing mode is used by the source operand, in the first instruction?
- Show contents of Flash Program Memory (mnemonics)
- Show contents of SRAM Data Memory, assuming variables are stored in sequential memory locations starting at address 0100_{16}.
- Modify the program to leave register r16 unchanged by making a copy (use r15).