Review: (Write on the board before class)
1) Multiple passes to simplify
2) Register allocation key in current optimizers
3) Fallacies: HW vs. SW: procedure call on VAX and i432

Today Ch 4: 4 examples of ISA (lecture 8)

Before general ideas if create own isntruction set
Next few lectures about popular instruction sets,

Who cares?
General information architect should know, guiding areas of investigation or areas not to investigate (back-of-envelope):
e.g., instruction prefetch unit that works fanatastic as long as straight-line sequences of 10 or more instructons (25% of VAX instructions branch)
e.g., fanastic speedup for string instructions in 80x86 computer that is going to run UNIX only

Detailed decisions:
e.g., if building a VAX, what is relative importance of each addressing mode to performance?

Notation: C + (how many haven't written a C program?)

$\leftarrow_n$ means transfer an n-bit quantity.

- A subscript is used to indicate selection of a bit from a field. Bits are labeled from the most significant bit starting at 0. The subscript may be a single digit (e.g., $R4_0$ yields the sign bit of R4) or a subrange (e.g., $R3_{24..31}$ yields the least significant byte of R3).
- A superscript is used to replicate a field (e.g., 0^{24} yields a field of zeros of length 24 bits).
- The variable `M` is used as an array that stands for main memory. The array is indexed by a byte address and may transfer any number of bytes. Ri are registers, usually 32-bits
- The symbol ## is used to concatenate two fields and may appear on either side of a data transfer.

$$R1_{0..15} \leftarrow_{16} \left(R2_0\right)^8 \#\# \; R3_{16..23}$$

Write rotate R1 right 4 bits?

Operand location: destination always on the left (no matter which convention used by manufacturer)

Size: Word = 32 bits, Halfword = 16bits, Byte = 8 bits, Double word = 64 bits

4 machines: VAX, IBM, 8086, DLX; each dominate a class of computing + new

VAX:
Start here because most other machines are subsets of the VAX
History 1970 PDP-11 16-bit address computer; 1975 crisis, running out of address bits; other 32-bit minicomputers; crash program for 32-bit mini => VAX

Q: How many already familiar with VAX?

Goals: PDP-11 customer base feel comfortable (PDP-11 data types, addressing modes, + emulation); simplify writing compilers and OS; make orthogonal architecture

4.2 *data types*

Variable length instruction encoding:
opcode 1 or 2
displacement: 8,16,32
operand specifiers: 8 bits, including register number
Examples 4.5

IBM 360: 1964
invented binary compatability : 7 different instruction sets before bet the company that it could be done; microcomputer to supercomputer running same instruction set

Goals: exploit storage hierarchy; support concurrent I/O; create a general purpose computer (business & scientific); compatability across family

IBM 370: 1970
update the instruction set
Goals: virtual memory, synchronization, remove data alignment requirement

1983: 370-XA, 24=> 31 bit addressing

1986: 3090 ESA/370: multiple address spaces

Registers: 16 32-bit GPR + 4 64-bit FPR + PC/PSW

Instruction format: 4.6

Break
Pick project: next Wednesday!! Have people raise hands of interest
How many have a project and partner; **HW 4.1, 4.5,4.8,4.11,4.12**
Bring envelope before class: Collect Check!
News: Demise of Gasset: didn't think money to be made in low-cost Mac (margins too low); left a hole in Mac market, Mac less atttractive because nothing equivalent to PC clones (pushed performance)

80x86: 10s of millions per year

INtel microprocessor in 1971:
Gordon Moore realized one more chance in about 1975; hired CS people; were ready in 1977; started crash effort for 16-bit micrcomputer; 8086

segmented addressing: 20 bit total

1978: Intel 8086; dedicated registers vs GP registers
1982: 80286 faster, plus protection
80386 in 1985: 32bit addressing + new registers
80486 in 1989: faster

Simple addressing modes
Empahsis on code density; 8 bit variable instructions

Look at figures of registers, addressing, examples

Bits	Data type	Our name	DEC's name
8	Integer	Byte	Byte
16	Integer	Halfword	Word
32	Integer	Word	Long
32	Floating point	Single precision	F_floating
64	Integer	Doubleword	Quad
64	Floating point	Double precision	D_floating or G_floating
128	Integer	Quadword	Octa
128	Floating point	Huge	H_floating
8n	Character string	Character	Character
4n	Binary coded decimal	Packed	Packed
8n	Numeric string	Unpacked	Numeric strings: Trailing and leading separate

FIGURE 4.2 VAX data types, their lengths, and names. The first letter of the DEC type (B, W, L, F, Q, D, G, O, H, C, P, T, S) is often used to complete an opcode name. As examples, the move opcodes include `MOVB`, `MOVW`, `MOVL`, `MOVF`, `MOVQ`, `MOVD`, `MOVG`, `MOVO`, `MOVH`, `MOVC3`, `MOVP`.

Example assembly instruction	Length	Meaning
MOVL @40(R4),30(R2)	5	M[M[40+R4]]←32 M[30+R2]
MOVAW R2,(R3)[R4]	4	R2←32 R3+(R4*2)
ADDL3 R5,(R6)+,(R6)+	4	t←M[R6];R6←R6+4; R5←t+M[R6]; R6←R6+4
CMPL -(R6),#10	7	R6←R6-4; *Set the condition code using: M[R6]-10*
CVTBW R10,(R8)	3	$R10_{16..31} \leftarrow_{16} (M[R8]_0)^8$ ## M[R8]
BEQL name	2	if equal(CC) {PC←name} *PC-128 ≤ name < PC+128*
BRW name	3	PC←name *PC-32768 ≤ name < PC+32768*
EXTZV (R8),R5,R6,-564(R7)	7	$t \leftarrow_{64}$ M[R7-564+(R5>>3)]; i←R5 & 31; j←R6 & 31; $M[R8] \leftarrow_{32} 0^{32-j}$ ## $t_{63-i-j+1..63-i}$;
MOVC3 @36(R9),(R10),35(R11)	6	R1←35+R11; R3←M[36+R4]; for (R0←M[R10];R0!=0;R0--) {M[R3]←8 M[R1]; R1++; R3++} R2=0; R4=0; R5=0
ADDD3 R0,R2,R4	4	(R0##R1)←64 (R2##R3)+(R4##R5) *register contents are type D floating point.*

FIGURE 4.5 Some examples of typical VAX instructions. VAX assembly language syntax puts the result operand last; we have put it first for consistency with other machines. Instruction length is given in bytes. The condition `equal(CC)` is true if the condition-code setting reflects equality after a compare. Remember that most instructions set the condition code; the only function of compare instructions is to set the condition code. The names `t`, `i`, `j` are used as a temporaries in the instruction descriptions; t is 64 bits in length, while i and j are 32 bits. The `EXTZV` instruction may appear mysterious. Its purpose is to extract a variable-length field and zero extend it to 32 bits. The source operands to the `EXTZV` are the starting bit position (which may be any distance from the starting byte address), the length of the field, and the word to extract the field from. The VAX numbers its bits from low order to high order, but we number bits in the reverse order. Thus, the subscripts adjust the bit offsets accordingly (which makes `EXTV` look more mysterious!). Although the result of the variable bit string operations are always 32 bits, the `MOVC3` changes the values of registers R0 through R5 as shown (although any of R0, R2, R4, and R5 could be used to hold the count).

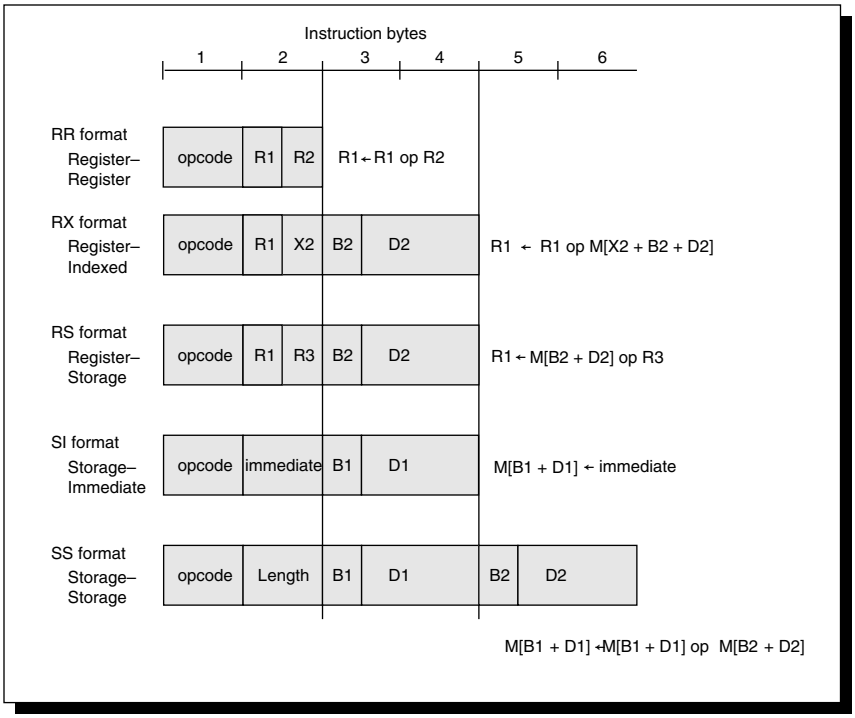

FIGURE 4.6 The 360/370 instruction formats. The possible instruction operands are a register (R1, R2, or R3), an 8-bit immediate, or a memory location. The opcode specifies where the operands reside and the addressing mode. The effective addresses for memory operands are formed using the sum of one or two registers (called B1, B2, or X2) and a 12-bit unsigned displacement field (called D1 or D2). In addition, the storage–storage instructions, which are all string oriented, specify an 8-bit length field. Other instruction formats have been added in later architectural extensions. These formats allowed the opcode space to be extended and new data types to be added. For loads, stores, and moves only one source operand is used and the operation only moves the data (see Figure 4.8 on page 152). For SS instructions, the length is one greater than the value in the instruction.

Type	Instruction example	Meaning
RR	`AR R4,R5`	`R4← R4+R5`
RX	`A R4,10(R5,R6)`	`R4← R4+M[R5+R6+10]`
RX	`BC Mask,20(R5,R6)`	`if (CC & Mask)!=0 {PC← 20+R5+R6}`
RS	`STM 20(R14),R2,R8`	`for(i=2;i<=8;i++) {M[R14+20+(i-2)*4] ←32 Ri}`
SI	`MVI 20(R5),#40`	`M[R5+20]←8 40`
SS	`MVC 10(R2),Len,20(R6)`	`for(i=0;i<Len+1;i++) {M[R2+10+i] ←8 M[R6+20+i]}`

FIGURE 4.7 Typical IBM 360 instructions with their meanings. The `MVC` instruction is shown with the length as the second operand. The length field is a constant in the instruction; standard 360 assembly language syntax includes the length with the first operand. The variable `i` used in the `MVC` and `STM` is a temporary.

Class	Register	Purposes of class or register
Data		**Used to hold and operate on data**
	AX	Used for multiply, divide, and I/O; sometimes an implicit operand; AH and AL also have dedicated uses in byte multiply, divide, decimal arithmetic
	BX	Can also be used as address-base register
	CX	Used for string operations and loop instructions; CL is the dynamic shift count
	DX	Used for multiply, divide, and I/O
Address		**Used to form 16-bit effective memory addresses (within segment)**
	SP	Stack pointer
	BP	Base register—used in based-addressing mode
	SI	Index register, and also used as string source base register
	DI	Index register, and also used as string destination base register
Segment		**Used to form 20-bit real memory addresses**
	CS	Code segment—used with instruction access
	SS	Stack segment—used for stack references (SP) or when BP is base register
	DS	Data segment—used when a reference is not for code (CS used), to the stack (SS used), or a string destination (ES used)
	ES	Extra segment—used when operand is string destination
Control		**Used for status and program control**
	IP	Instruction pointer—provides the offset of the currently executing instruction (this is the lower 16-bits of the effective PC)
	FLAGS	Contains six condition code bits—carry, zero, sign, borrow, parity, and overflow—and three status control bits

FIGURE 4.9 The 14 registers on the 8086. The table divides them into four classes that have restricted uses. In addition, many of the individual registers are required for certain instructions. The data registers have an upper and lower half: *x*L refers to lower byte and *x*H to upper byte of register *x.*

Source/destination operand type	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

FIGURE 4.10 Instruction types for the arithmetic, logical, and data-transfer instructions. The 8086 allows the combinations shown. Immediates may be 8 or 16 bits in length; a register is any one of the 12 major registers in Figure 4.9 (not one of the control registers). The only restriction is the absence of memory–memory mode.

Instruction	Function
JE name	if equal(CC) {IP←name}; PC–128 ≤ name < PC+128
JMP name	IP←name
CALLF name,seg	SP←SP–2; M[SS:SP]←IP+5; SP←SP–2; M[SS:SP]←CS; IP←name; CS←seg;
MOVW BX,[DI+45]	BX←$_{16}$M[DS:DI+45]
PUSH SI	SP←SP–2; M[SS:SP]←SI
POP DI	DI←M[SS:SP]; SP←SP+2
ADD AX,#6765	AX←AX+6765
SHL BX,1	BX←$BX_{1..15}$ ## 0
TEST DX,#42	Set CC flags with DX & 42
MOVSB	M[ES:DI]←$_{8}$M[DS:SI]; DI←DI+1; SI←SI+1

FIGURE 4.11 Some typical 8086 instructions and their functions. A list of the most frequent operations appears in Figure 4.12 (page 158). We use the abbreviation SR:X to indicate the formation of an address with segment register SR and offset X. This effective address corresponding to SR:X is (SR<<4)+X. The `CALLF` saves the IP of the next instruction and the current CS on the stack.

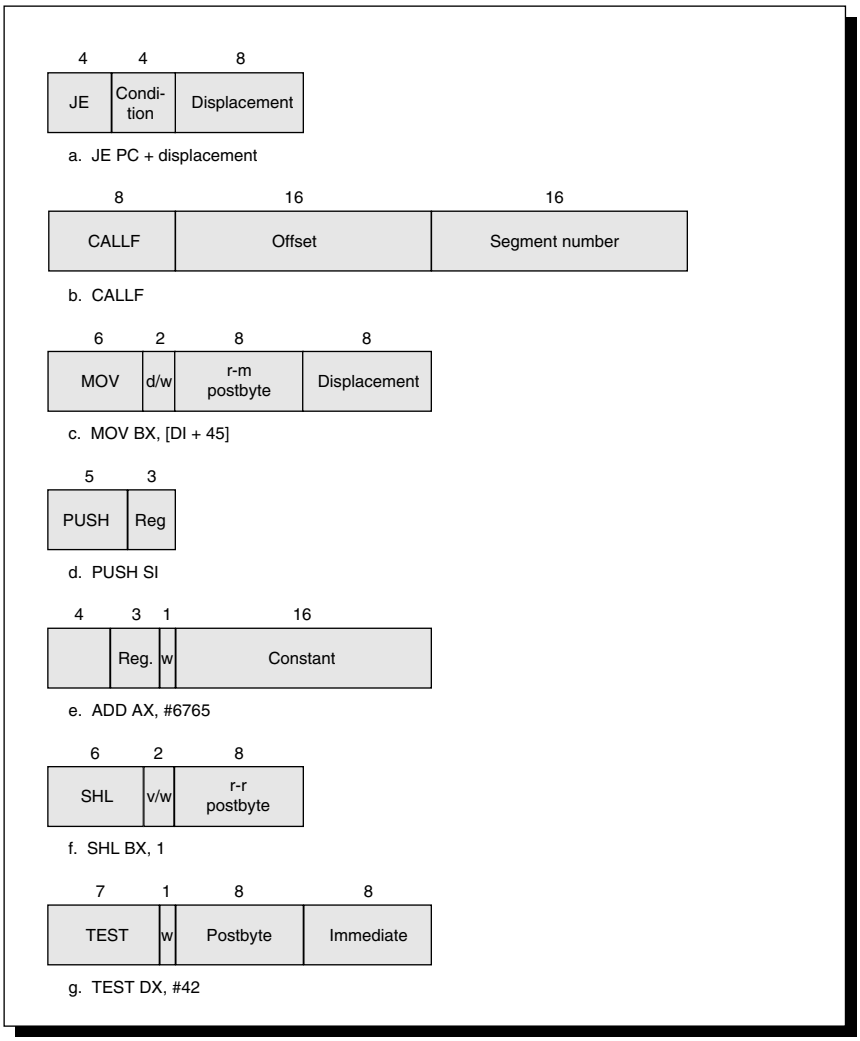

FIGURE 4.13 Typical 8086 instruction formats. The encoding of the postbyte is shown in Figure 4.14. Many instructions contain the 1-bit field w, which says whether the operation is a byte or word. Fields of the form v/w or d/w are a d-field or v-field followed by the w-field. The d-field in `MOV` is used in instructions that may move to or from memory and shows the direction of the move. The field v in the `SHL` instruction indicates a variable-length shift; variable-length shifts use a register to hold the shift count. The `ADD` instruction shows a typical optimized short encoding usable only when the first operand is AX. Overall instructions may vary from one to six bytes in length.

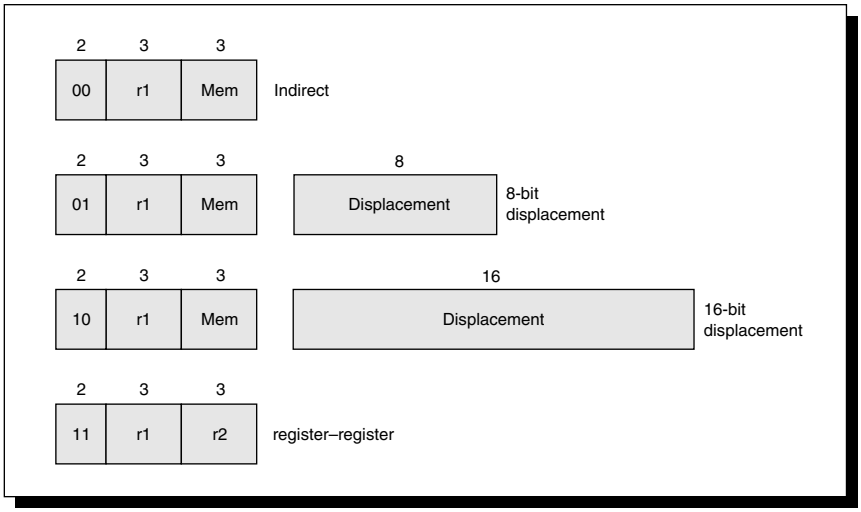

FIGURE 4.14 There are four postbyte encodings on the 8086 designated by a 2-bit tag. The first three indicate a register–memory instruction, where Mem is the base register. The fourth form is register–register.

Tools

P1. Instruction statistics tools for 370 or 80x86. (Exercise 4.22)

P2. Link-time instrumentation for cache simulation.

P3. Improving DLX tools.

P4. DLX instruction scheduler (Exercise 6.22)

P5. Adding vector instructions to the DLX simulator (Exercise 7.13)

P6. Adding vector checking to the DLX compiler (Exercise 7.14)

Input/Output

P7 Collecting statistics on disk arm motion

P8 Measuring I/O patterns in supercomputers

P9 I/O overhead experiments

P10 I/O performance experiments (Exercise 9.15)

P11 Generalized scaling formulas

Cost and Performance

P12 Evaluate a "stall" cache

P13 Distributed system cost

P14 Benchmarking Cray instruction set architecture vs. newer architectures

P15 Generalized Benchmarks

P16 Synthetic SPEC benchmark