CS271

- **COMPUTER ARCHITECTURE AND ASSEMBLY LANGUAGE**
  - **Bruce D'Ambrosio - 107 Dearborn, 7-5563**
    - **dambrosi@cs.orst.edu**
  - **Arvind Guruprasad, guruprar@cs.orst.edu**
  - **Text: Tannenabum, Structured Computer Org. 4 ed.**

Syllabus

- **Functional organization of digital computers.**
  - **Components, Logic-level, MicroArchitecture, ISA**
- **OS & Assembly language**
  - **addressing, stacks, argument passing, arithmetic operations, decisions, macros, modularization, linkers and debuggers.**
- **Prereq: CS 161, MTH 231**
- **Written Homework: weekly, 10%**
- **Programming: 10% (2 assignments ?)**
- **2 Quizzes: 5% each**
- **2 Midterms: 15% each**
- **Final: 40%**

Schedule

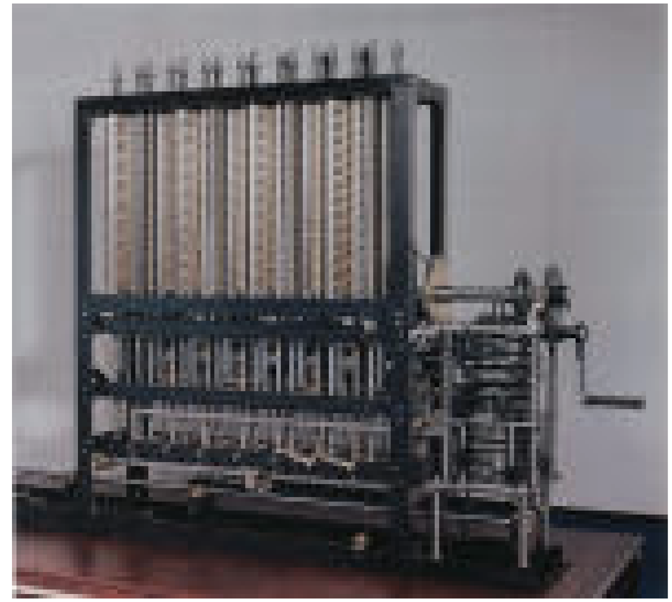
Week	Chapter	Notes
1	1	Quiz
2	2	
3	3.1 - 3.4	
4	3.5, 4.1	Midterm 1
5	4.2 - 4.7	
6	5.1 - 5.5	
7	5.6 - 5.7	Midterm 2
8	6.1 - 6.3	Holiday
9	6.4 - 6.5, 7.1	
10	7	

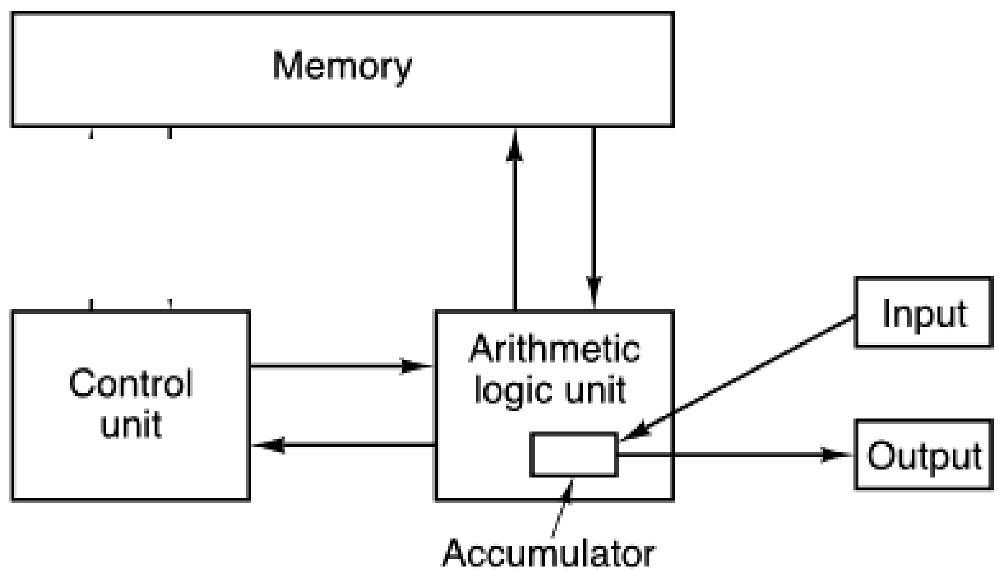
History

Year	Name	Made by	Comments
1834	Analytical Engine	Babbage	First attempt to build a digital computer
1936	Z1	Zuse	First working relay calculating machine
1943	COLOSSUS	British gov't	First electronic computer
1944	Mark I	Aiken	First American general-purpose computer
1946	ENIAC I	Eckert/Mauchley	Modern computer history starts here
1949	EDSAC	Wilkes	First stored-program computer
1951	Whirlwind I	M.I.T.	First real-time computer
1952	IAS	Von Neumann	Most current machines use this design
1960	PDP-1	DEC	First minicomputer (50 sold)
1961	1401	IBM	Enormously popular small business machine
1962	7094	IBM	Dominated scientific computing in the early 1960s
1963	B5000	Burroughs	First machine designed for a high-level language
1964	360	IBM	First product line designed as a family
1964	6600	CDC	First scientific supercomputer
1965	PDP-8	DEC	First mass-market minicomputer (50,000 sold)
1970	PDP-11	DEC	Dominated minicomputers in the 1970s
1974	8080	Intel	First general-purpose 8-bit computer on a chip
1974	CRAY-1	Cray	First vector supercomputer
1978	VAX	DEC	First 32-bit superminicomputer
1981	IBM PC	IBM	Started the modern personal computer era
1985	MIPS	MIPS	First commercial RISC machine
1987	SPARC	Sun	First SPARC-based RISC workstation
1990	RS6000	IBM	First superscalar machine

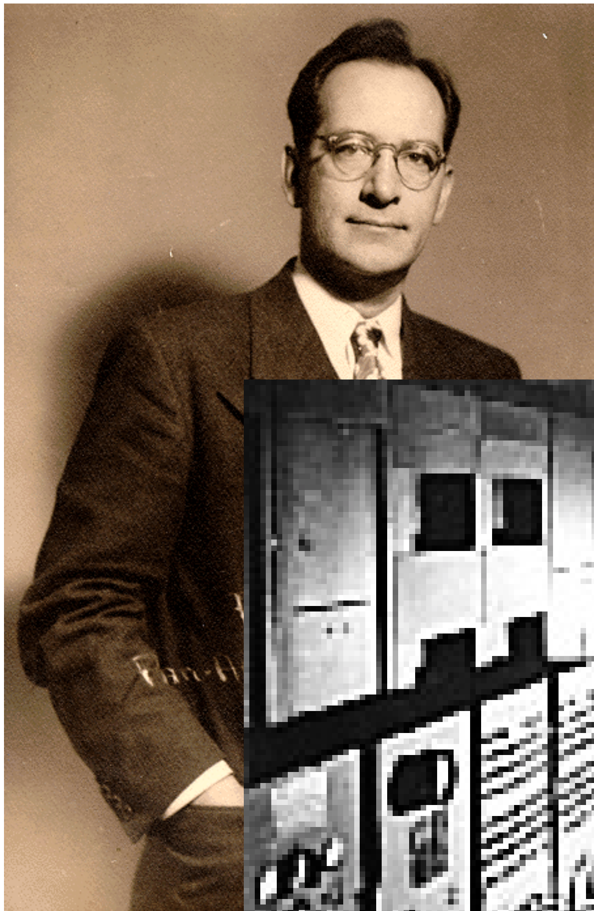
The Difference Engine

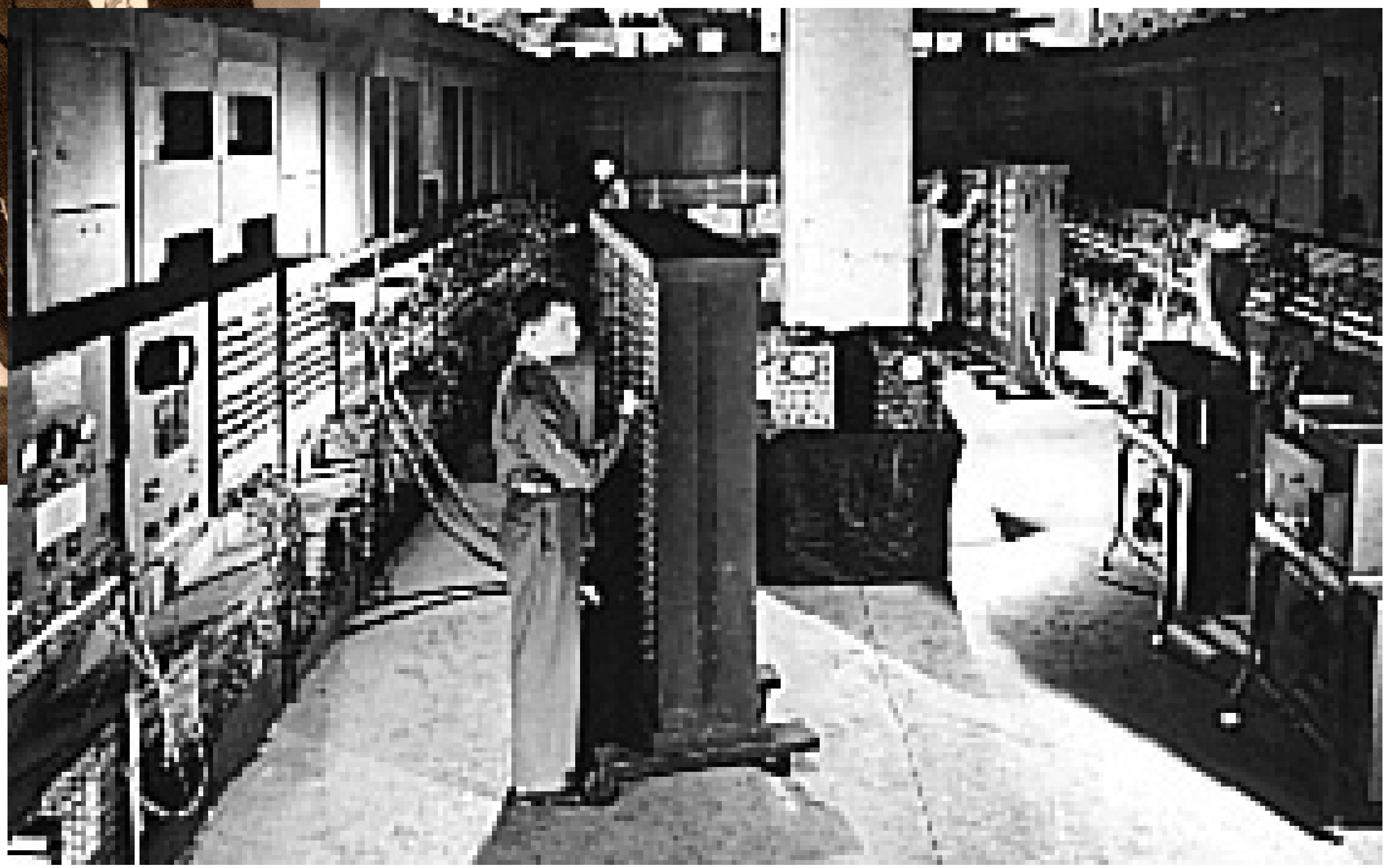
Ada Lovelace



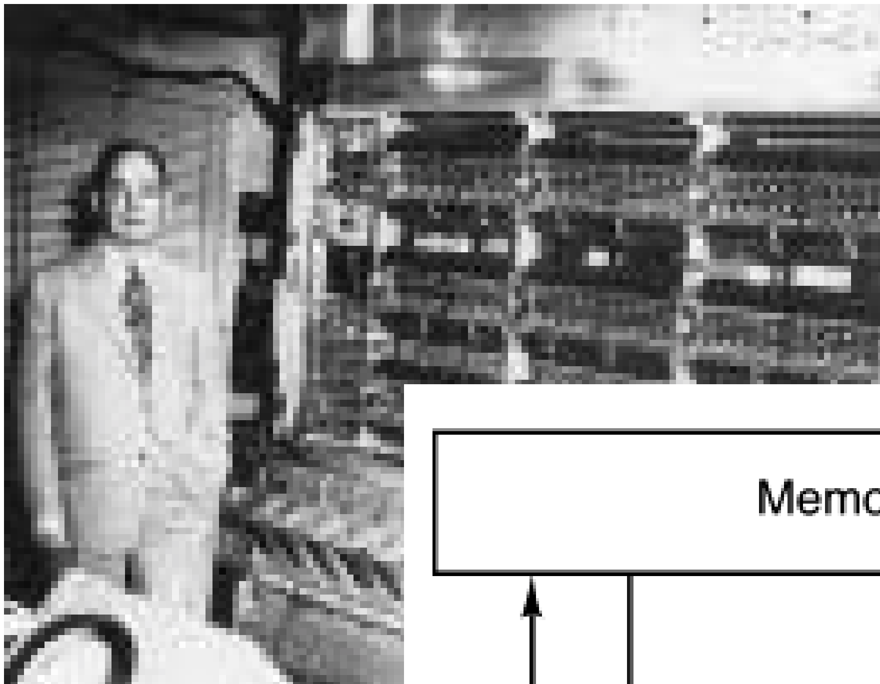


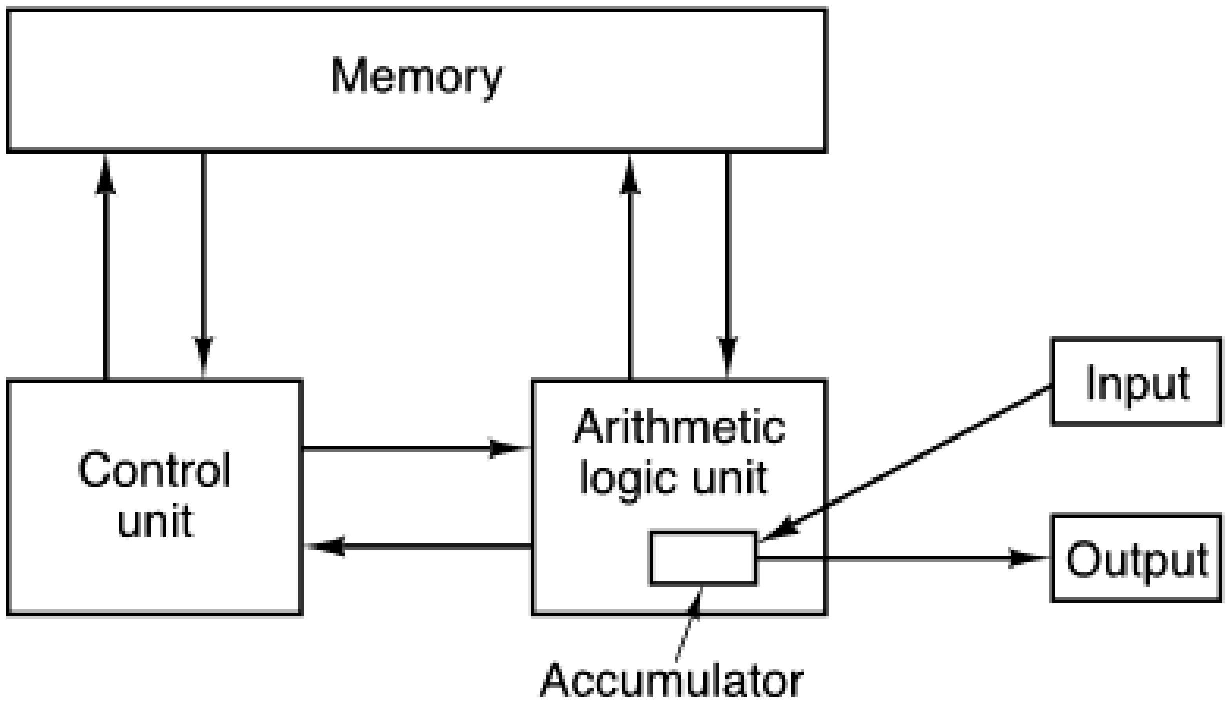
ENIAC





Von Neumann and IAS



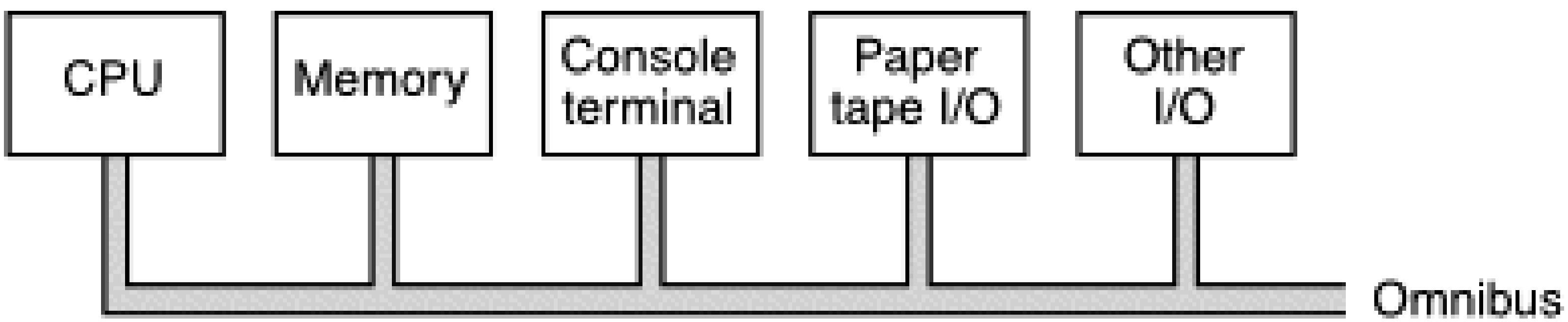


IBM - 1960s

1401

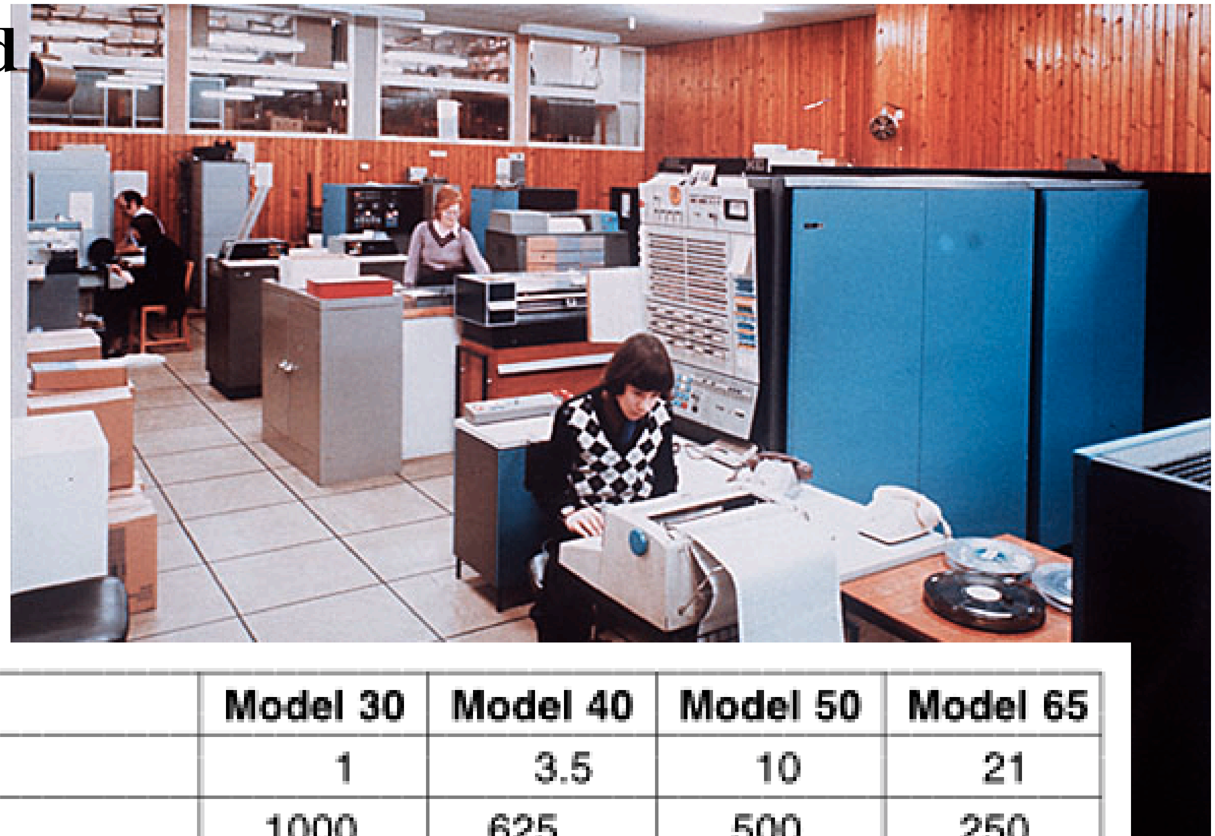
7094

PDP-8



IBM 360

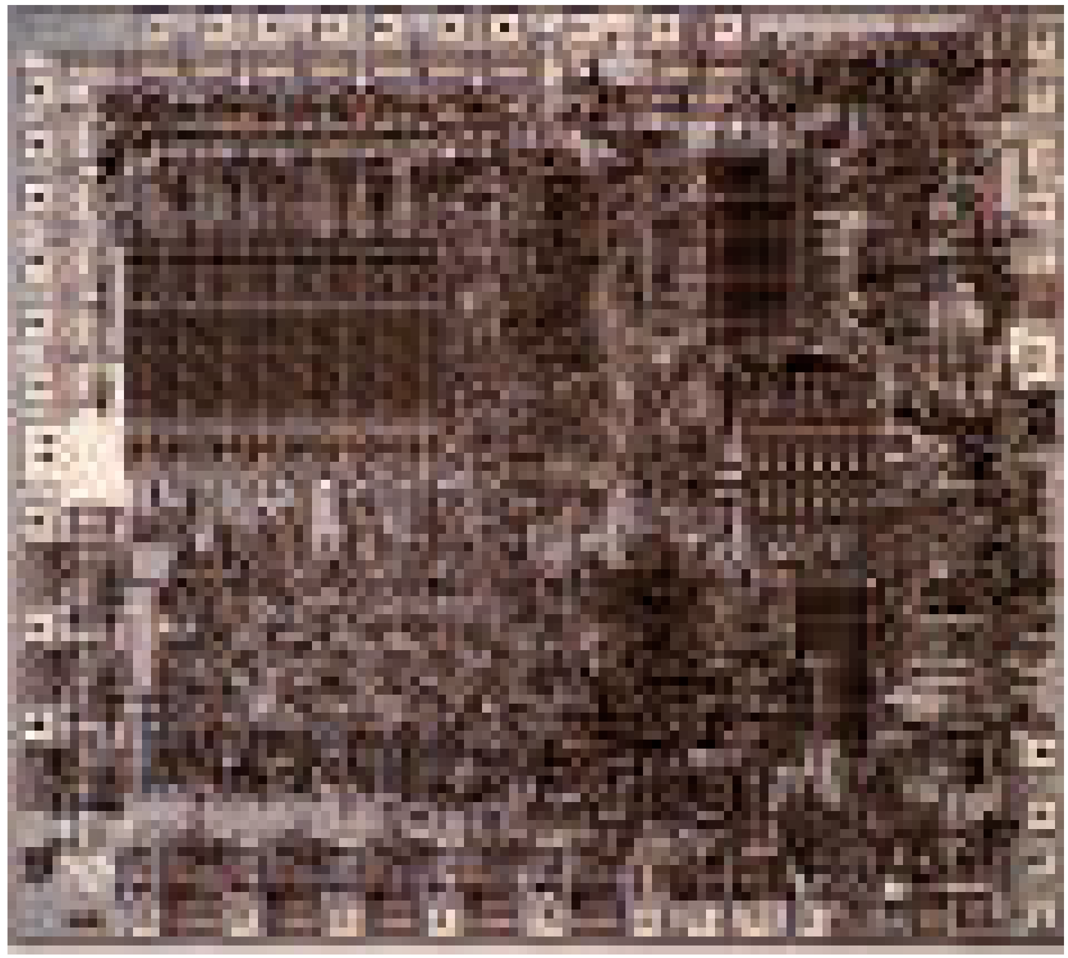
- Microprogrammed
- Family
- Multiprogram



Property	Model 30	Model 40	Model 50	Model 65
Relative performance	1	3.5	10	21
Cycle time (nsec)	1000	625	500	250
Maximum memory (KB)	64	256	256	512
Bytes fetched per cycle	1	2	4	16
Maximum number of data channels	3	3	4	6

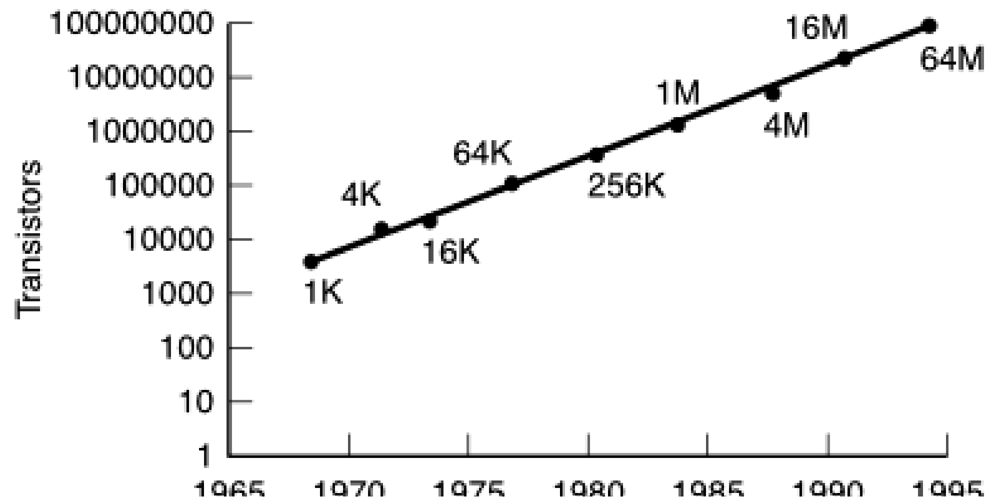
The VLSI revolution

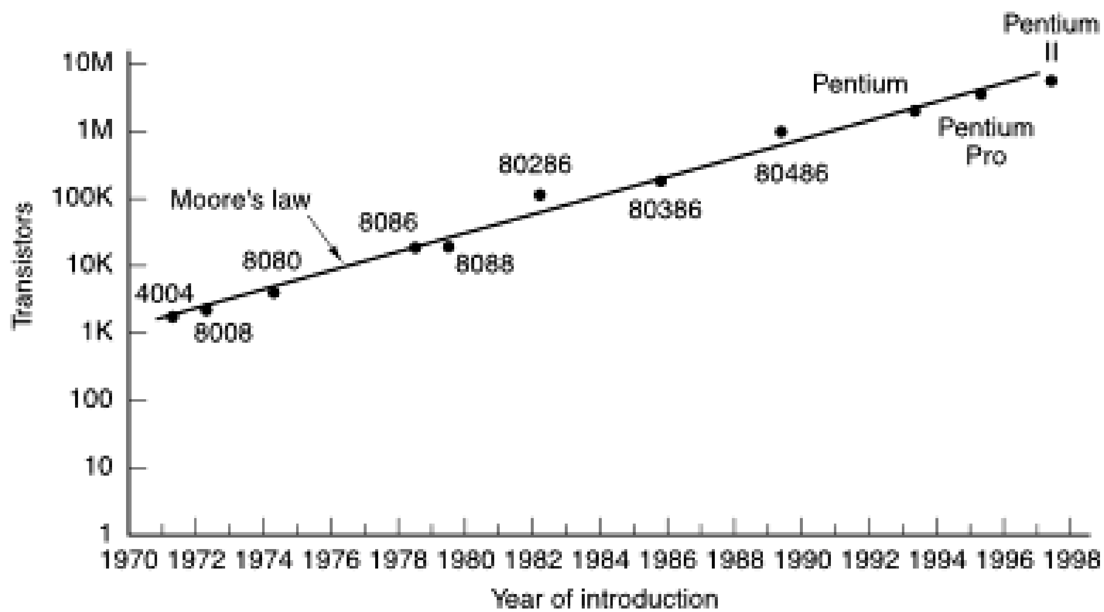
- **4004**
- **8008**
- **8080**
- **8088**
- **8086**
- **80286**
- **80386**
- **80486**
- **Pentium**
- **PII**
- **PIII**
- **PIV**
- **…**



Exponential Growth

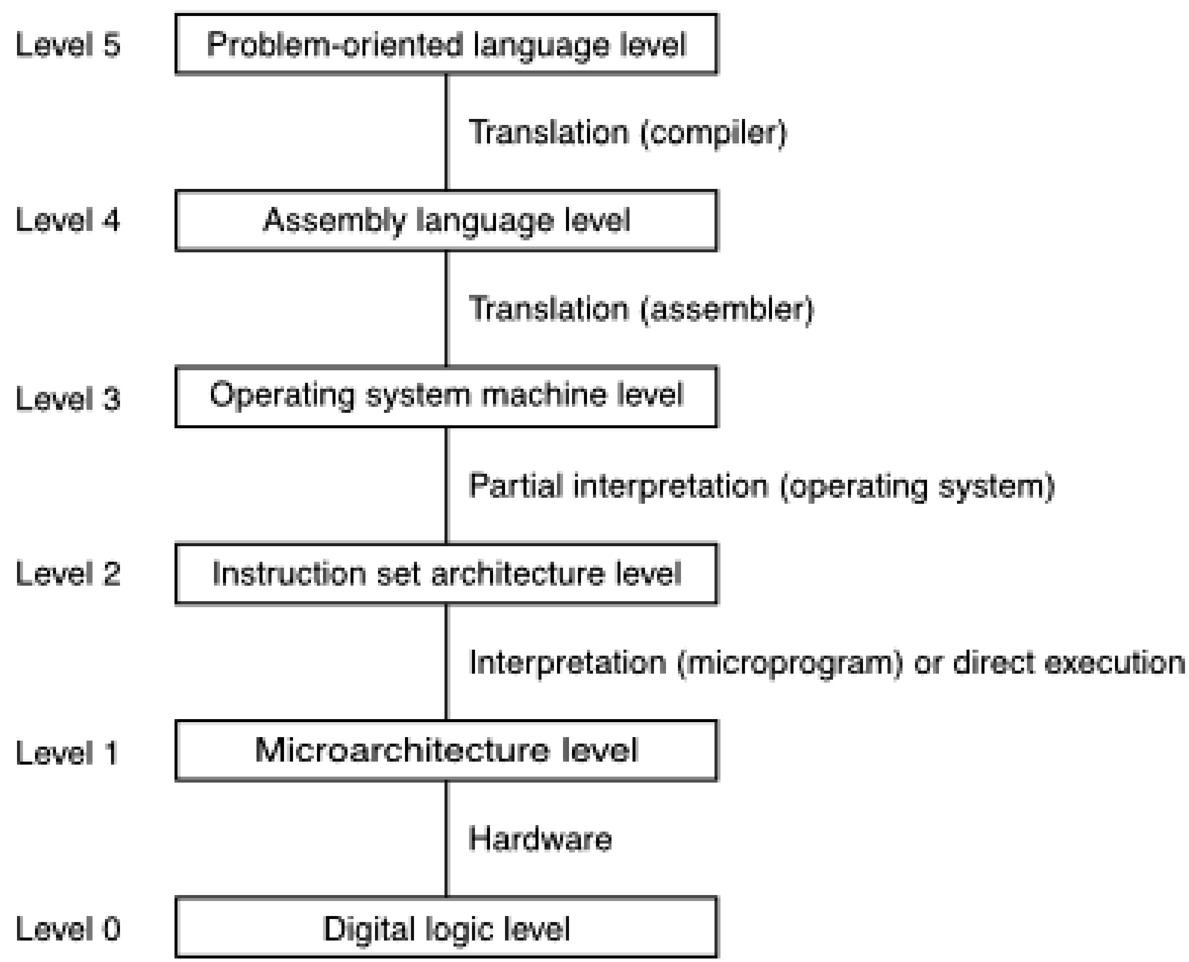
- Machine code
- Single-thread OS
- Multi-thread
- On-line
- Graphical UI
- Multi-media
- Virtual Reality (?)



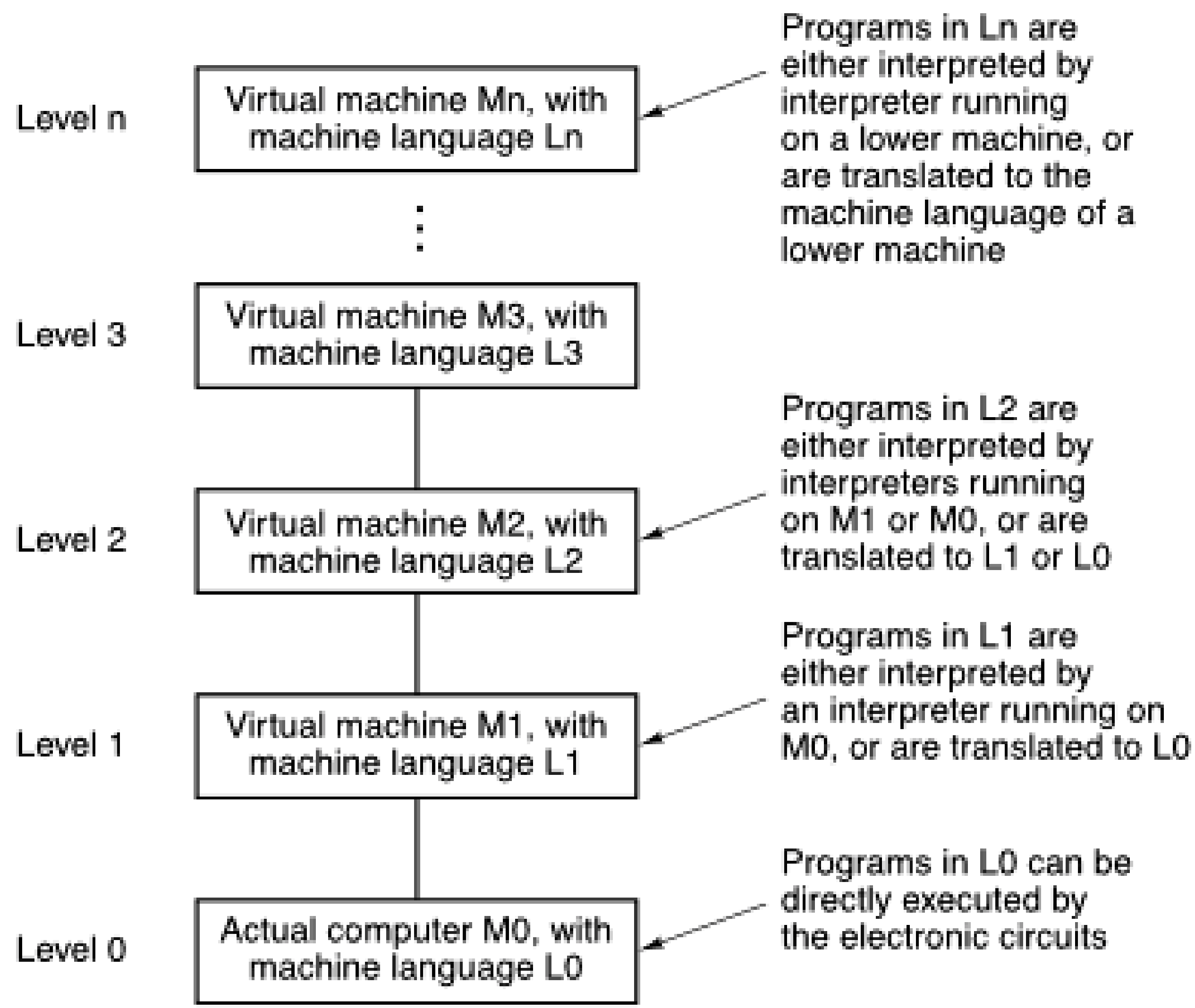


Computer Organization

The layered model

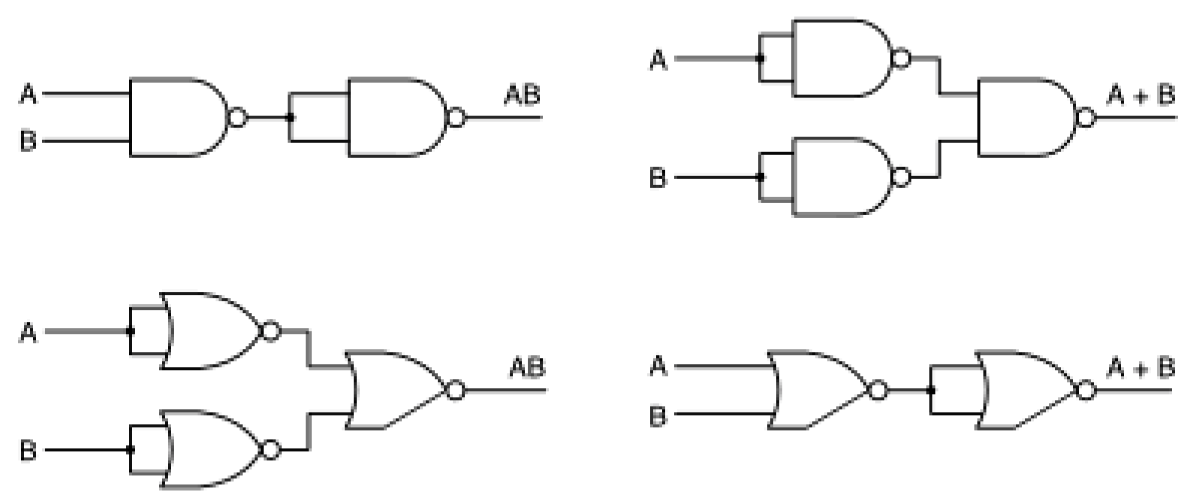


Layered Machine model



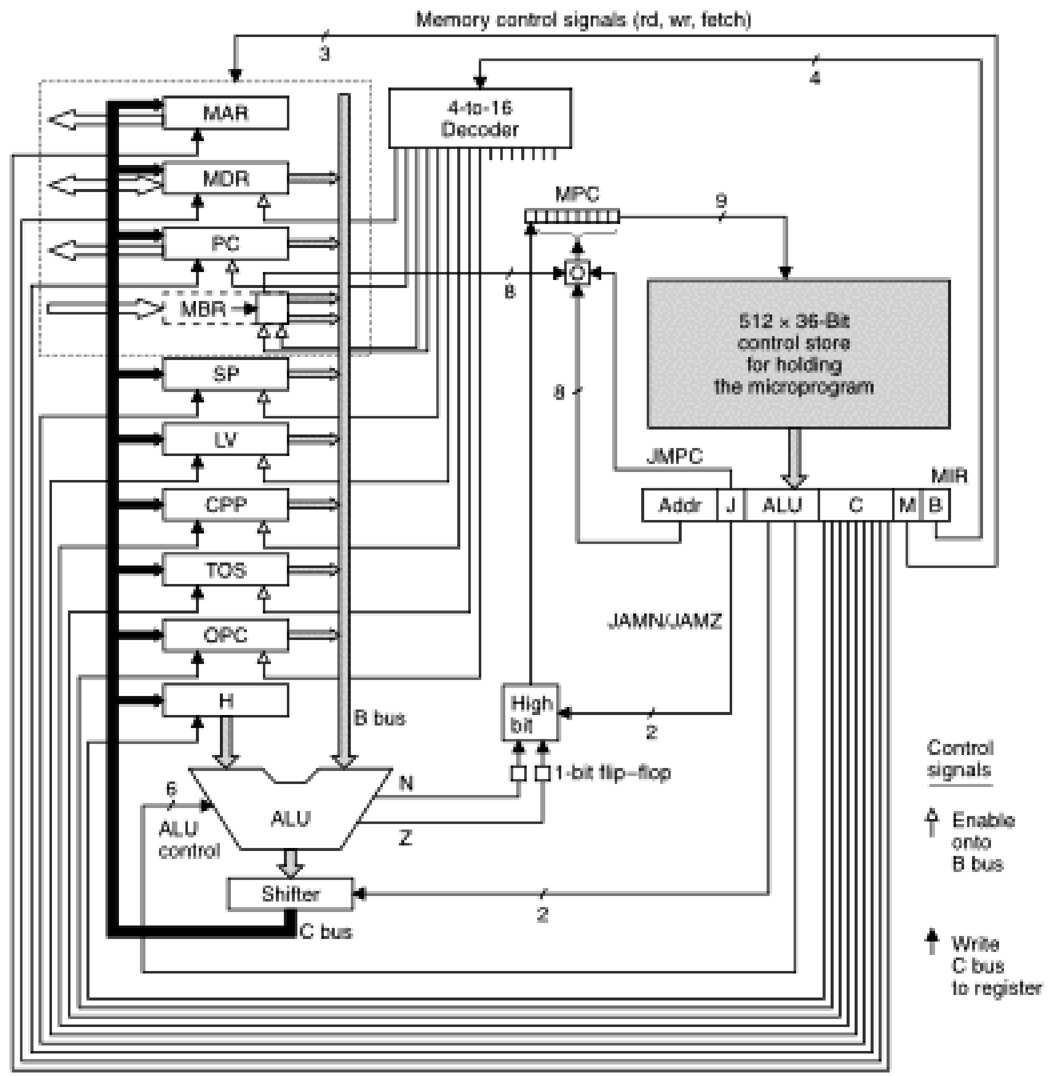
Digital Logic Level

- Logical Abstraction of physical hardware
- Basic elements are gates.
- How are gates composed?



MicroArchitecture

- Registers
- Datapaths
- Functional Units
  - ALU
- First view of entire machine - sort of



Instruction Set Architecture

- **Instruction-visible registers**
- **Instruction-view of storage**
- **Instruction-available operations**
- **Instruction word format**

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOV DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract DST from SRC
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract DST & carry from SRC
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

Test/compare

TST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT ADDR	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source　　# = shift/rotate count
DST = destination　　LV = # locals

ISA View of CPU (Cont’d)

Register	Alt. name	Function
R0	G0	Hardwired to 0. Stores into it are just ignored.
R1 – R7	G1 – G7	Holds global variables
R8 – R13	O0 – O5	Holds parameters to the procedure being called
R14	SP	Stack pointer
R15	O7	Scratch register
R16 – R23	L0 – L7	Holds local variables for the current procedure
R24 – R29	I0 – I5	Holds incoming parameters
R30	FP	Pointer to the base of the current stack frame
R31	I7	Holds return address for the current procedure

Figure 5-4. The UltraSPARC II’s general registers.

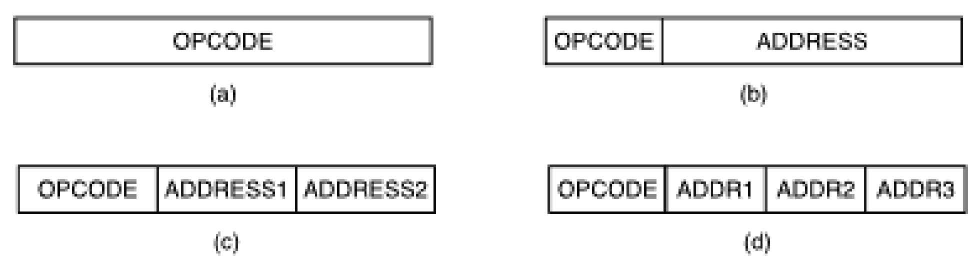


Figure 5-9. Four common instruction formats: (a) Zero-

Type	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Signed integer	×	×	×	×	
Unsigned integer					
Binary coded decimal integer					
Floating point			×	×	

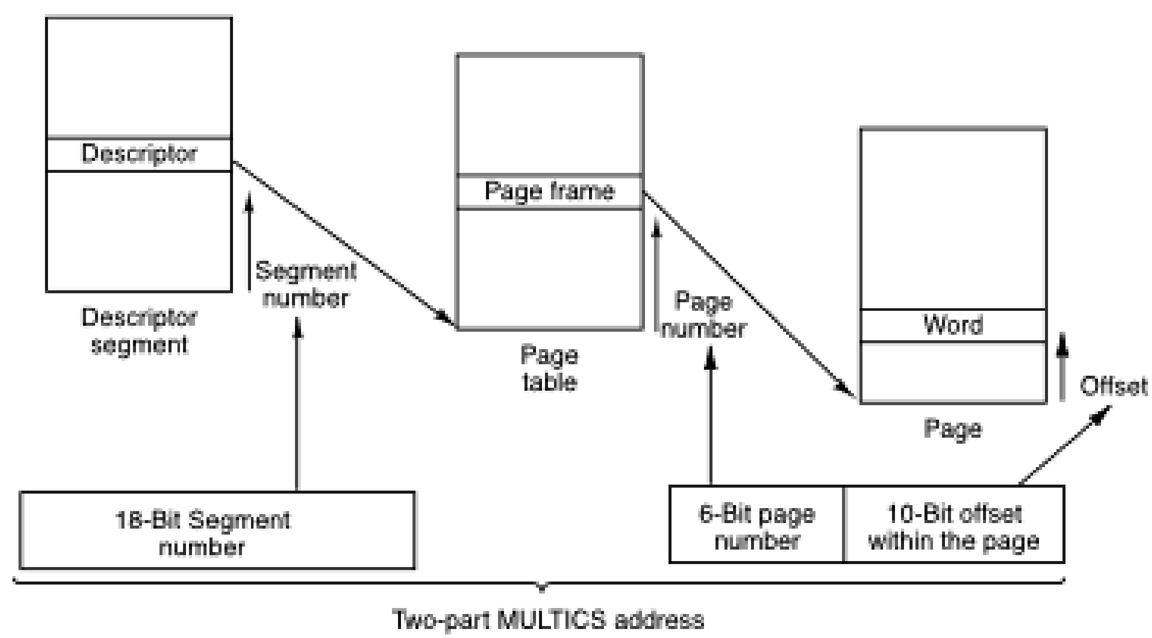
Figure 5-8. The JVM numeric data types.

Operating System Level

- Storage management
  - Memory
  - Files
- Programs
- Processes
- Communication

Category	Some examples
File management	Open, read, write, close, and lock files
Directory management	Create and delete directories; move files around
Process management	Spawn, terminate, trace, and signal processes
Memory management	Share memory among processes; protect pages
Getting/setting parameters	Get user, group, process ID; set priority
Dates and times	Set file access times; use interval timer; profile execution
Networking	Establish/accept connection; send/receive message
Miscellaneous	Enable accounting; manipulate disk quotas; reboot the system

Figure 6-29. A rough breakdown of the UNIX system calls.



Assembly Language Level

- **Symbolic language**
  - **Keywords**
  - **Name management**
  - **Pseudo-Operations**
- **Meta-language**
  - **macros**

```
000010
100001
111001
001100

Limit: WORD 2
A:     ADD R1, R0
       CMP R0, Limit
       BLT A
```

Item	Macro call	Procedure call
When is the call made?	During assembly	During execution
Is the body inserted into the object program every place the call is made?	Yes	No
Is a procedure call instruction inserted into the object program and later executed?	No	Yes
Must a return instruction be used after the call is done?	No	Yes
How many copies of the body appear in the object program?	One per macro call	1

Basics of software development

- **Program development environment**
  - Assembler
  - Linker
  - Loader
  - Debugger

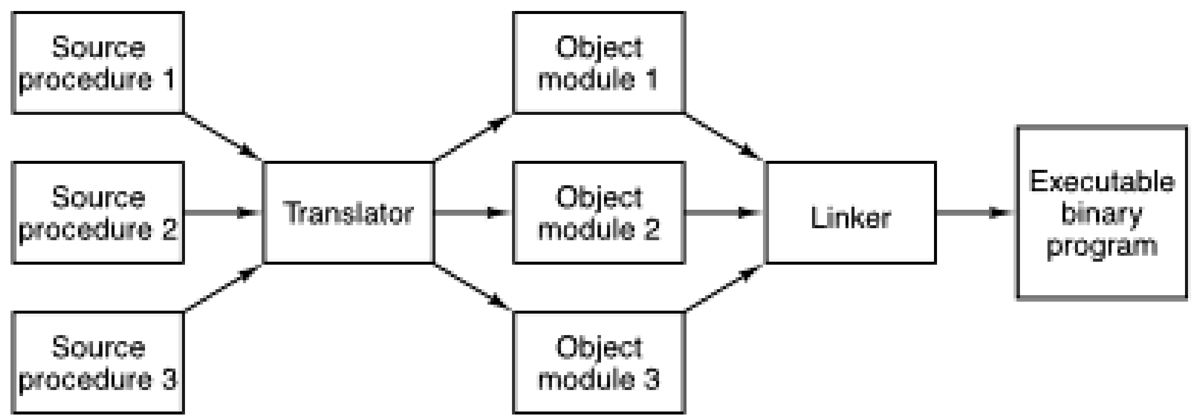


Figure 7-13. Generation of an executable binary program from a collection of independently translated source procedures requires using a linker.