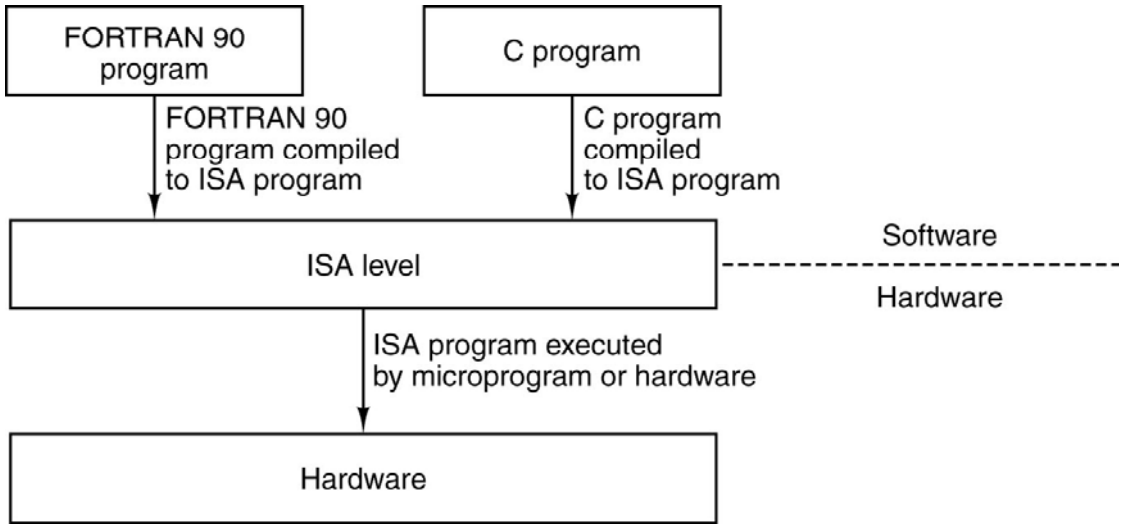
The Instruction Set Architecture Level

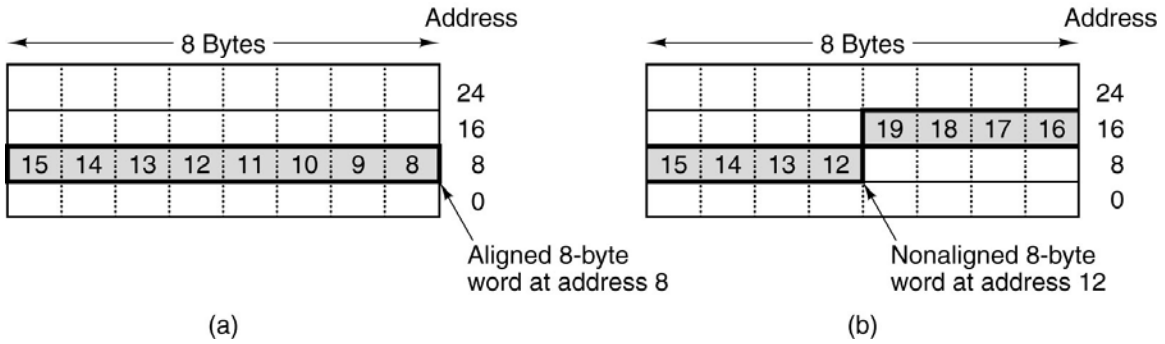
Chapter 5

ISA Level



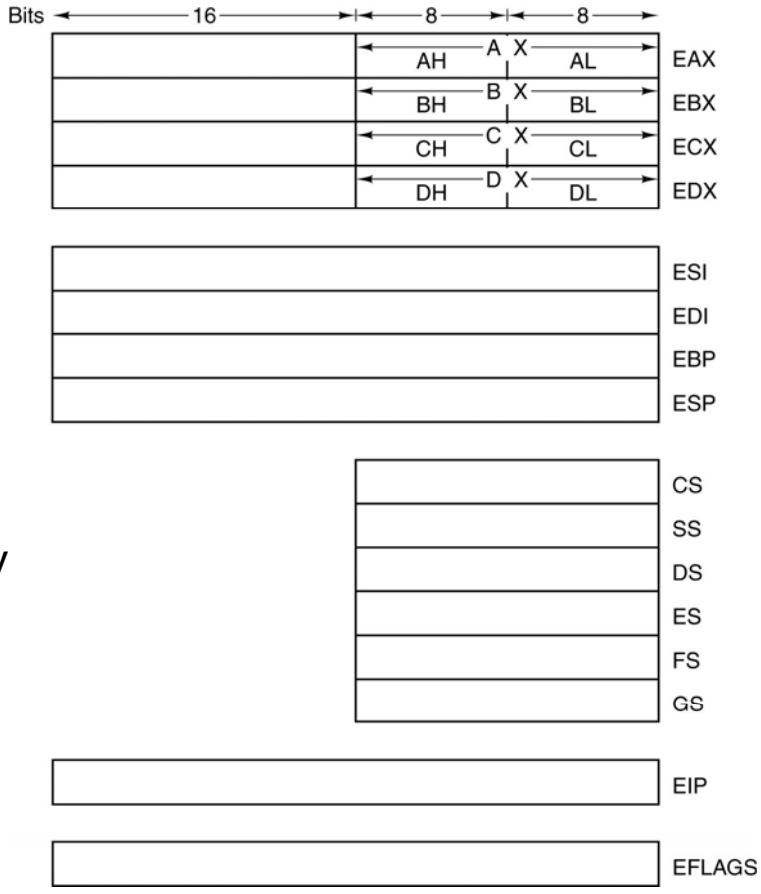
The ISA level is the interface between the compilers and the hardware.

Memory Models



An 8-byte word in a little-endian memory. (a) Aligned. (b) Not aligned. Some machines require that words in memory be aligned.

Overview of the Pentium 4 ISA Level



The Pentium 4's primary registers.

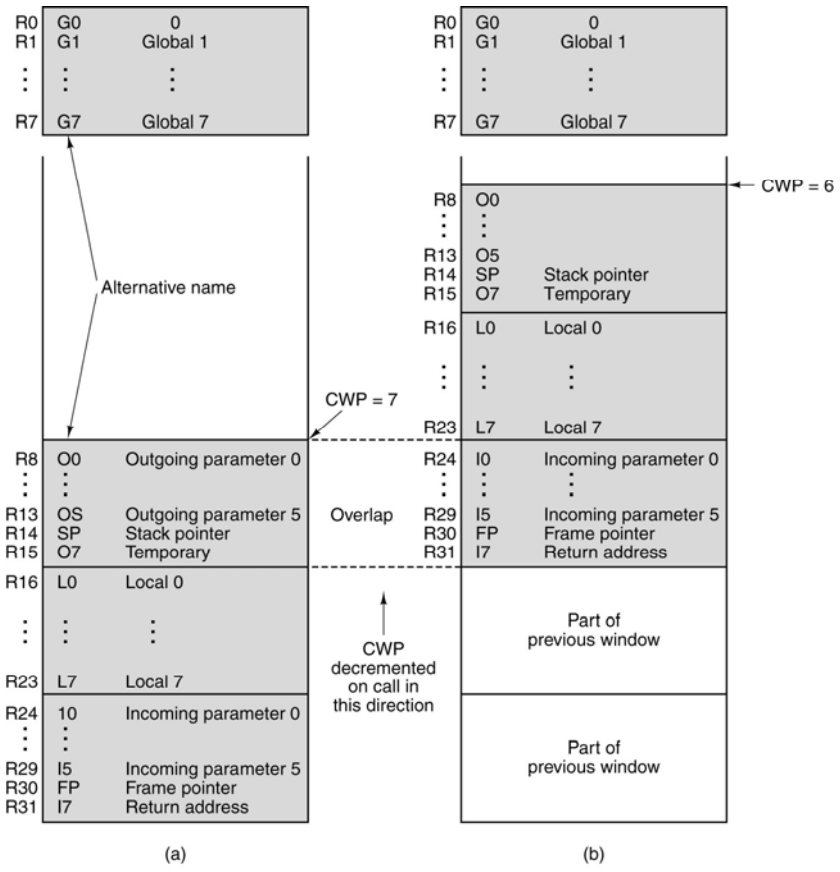
Overview of the UltraSPARC III ISA Level (1)

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

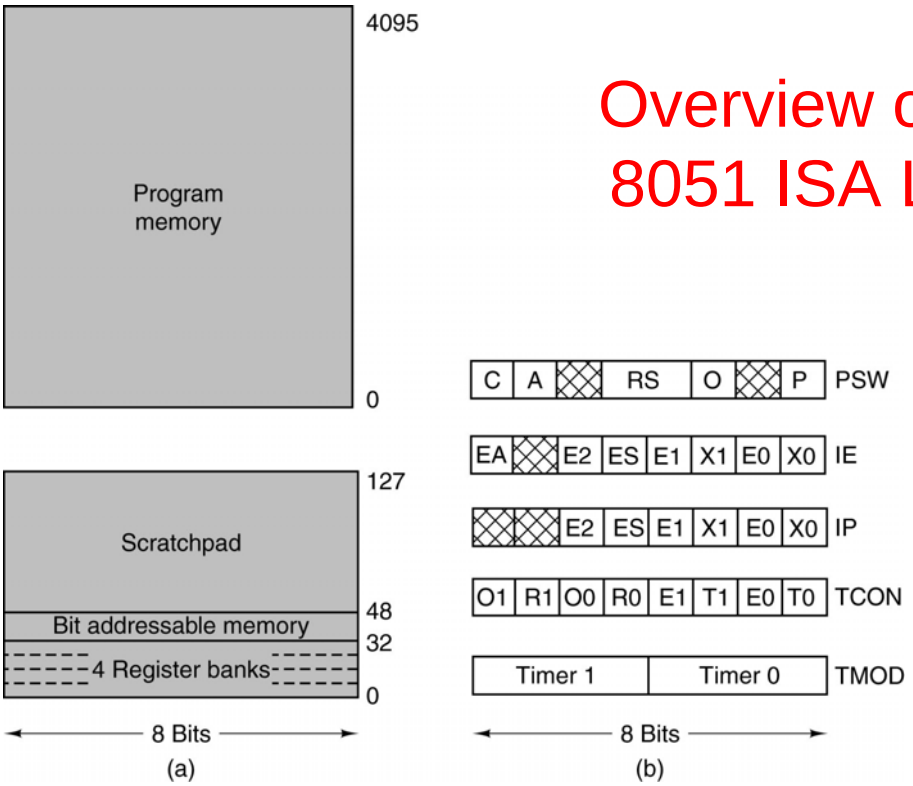
The UltraSPARC III's general registers.

Overview of the UltraSPARC III ISA Level (2)

Operation of the UltraSPARC III register windows.



Overview of the 8051 ISA Level



(a) On-chip memory organization for the 8051.
(b) Major 8051 registers.

Data Types on the Pentium 4

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×		
Unsigned integer		×	×	×		
Binary coded decimal integer		×				
Floating point				×	×	

The Pentium 4 numeric data types.
Supported types are marked with ×.

Data Types on the UltraSPARC III

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit						
Signed integer		×	×	×	×	
Unsigned integer		×	×	×	×	
Binary coded decimal integer						
Floating point				×	×	×

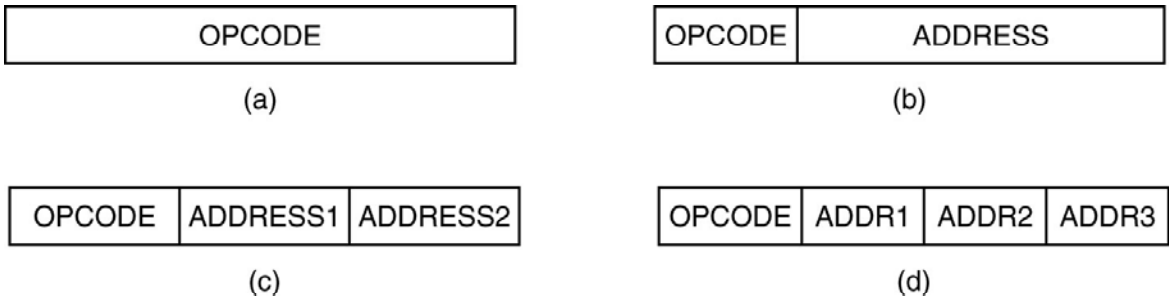
The UltraSPARC III numeric data types.
Supported types are marked with ×.

Data Types on the 8051

Type	1 Bit	8 Bits	16 Bits	32 Bits	64 Bits	128 Bits
Bit	×					
Signed integer		×				
Unsigned integer						
Binary coded decimal integer						
Floating point						

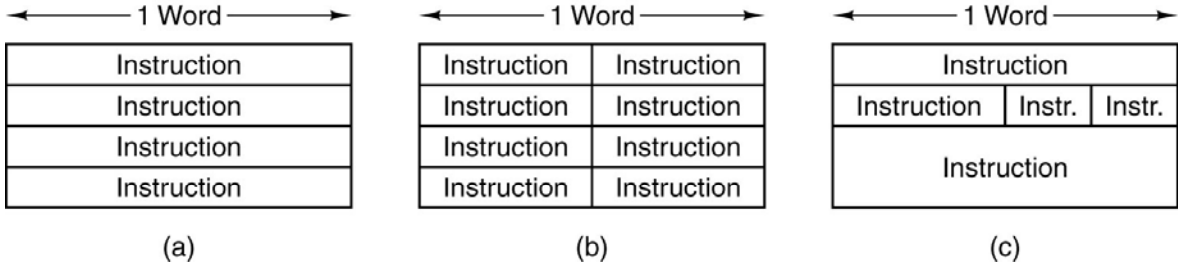
The 8051 numeric data types.
Supported types are marked with ×.

Instruction Formats (1)



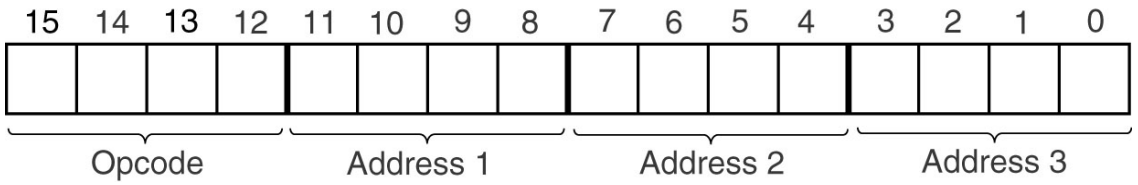
Four common instruction formats:
(a) Zero-address instruction. (b) One-address instruction
(c) Two-address instruction. (d) Three-address instruction.

Instruction Formats (2)



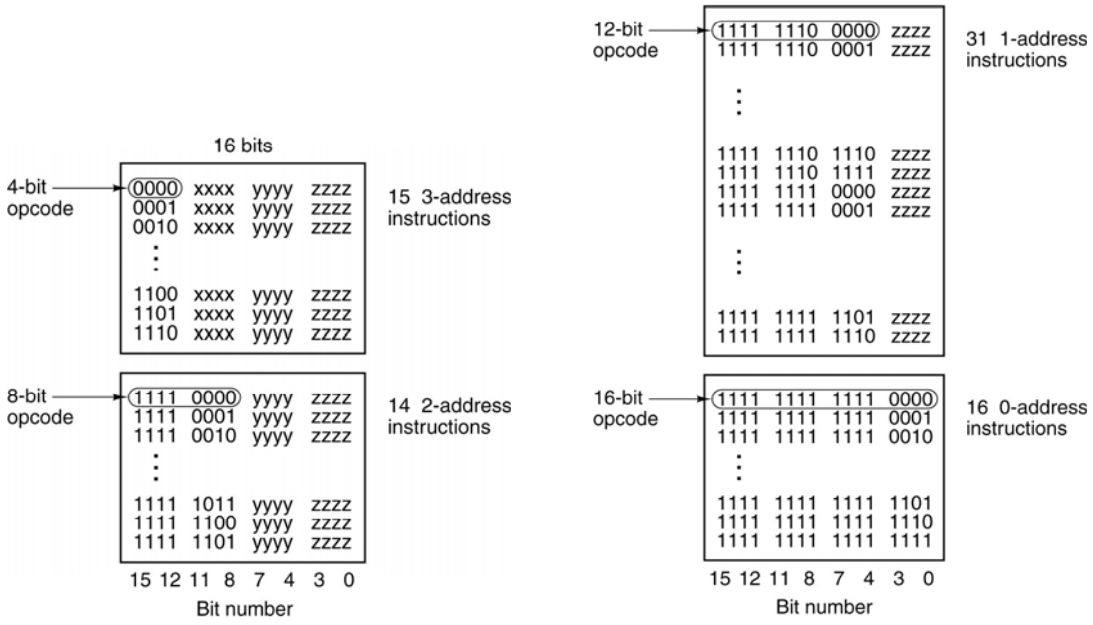
Some possible relationships between instruction and word length.

Expanding Opcodes (1)



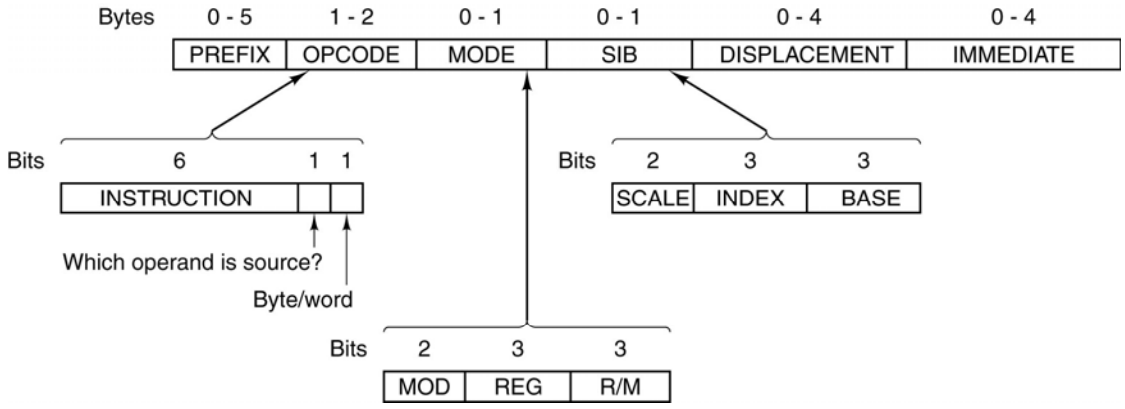
An instruction with a 4-bit opcode and three 4-bit address fields.

Expanding Opcodes (2)



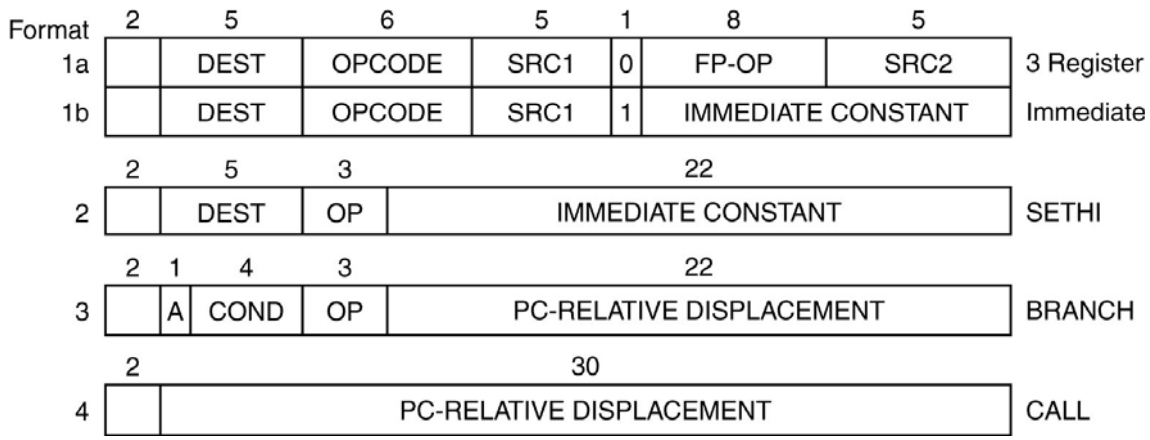
An expanding opcode allowing 15 three-address instructions, 14 two-address instructions, 31 one-address instructions, and 16 zero-address instructions. The fields marked *xxxx*, *yyyy*, and *zzzz* are 4-bit address fields.

The Pentium 4 Instruction Formats



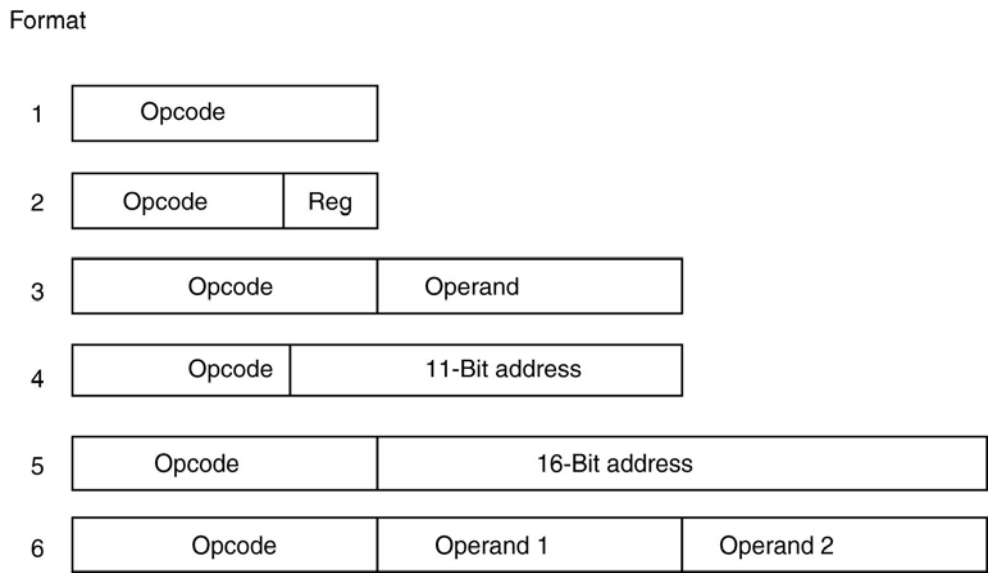
The Pentium 4 instruction formats.

The UltraSPARC III Instruction Formats



The original SPARC instruction formats.

The 8051 Instruction Formats



The 8051 instruction formats.

Addressing

MOV	R1	4

An immediate instruction for loading 4 into register 1.

```
        MOV R1,#0          ; accumulate the sum in R1, initially 0
        MOV R2,#A          ; R2 = address of the array A
        MOV R3,#A+4096     ; R3 = address of the first word beyond A
LOOP:   ADD R1,(R2)        ; register indirect through R2 to get operand
        ADD R2,#4          ; increment R2 by one word (4 bytes)
        CMP R2,R3          ; are we done yet?
        BLT LOOP           ; if R2 < R3, we are not done, so continue
```

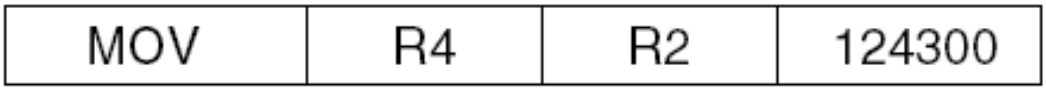
Register Indirect Addressing: a generic assembly program for computing the sum of the elements of an array.

Indexed Addressing (1)

```
        MOV R1,#0          ; accumulate the OR in R1, initially 0
        MOV R2,#0          ; R2 = index, i, of current product: A[i] AND B[i]
        MOV R3,#4096       ; R3 = first index value not to use
LOOP:   MOV R4,A(R2)       ; R4 = A[i]
        AND R4,B(R2)       ; R4 = A[i] AND B[i]
        OR R1,R4           ; OR all the Boolean products into R1
        ADD R2,#4          ; i = i + 4 (step in units of 1 word = 4 bytes)
        CMP R2,R3          ; are we done yet?
        BLT LOOP           ; if R2 < R3, we are not done, so continue
```

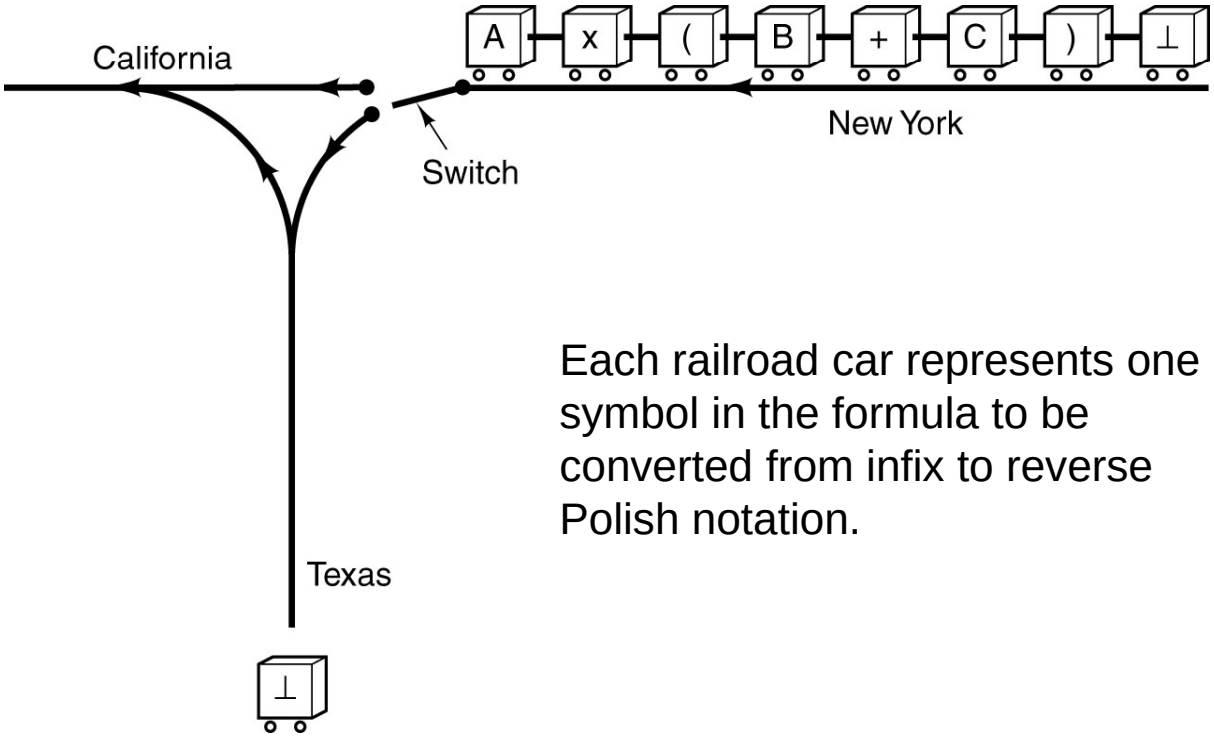
A generic assembly program for computing the OR of *Ai* AND *Bi* for two 1024-element arrays.

Indexed Addressing (2)



A possible representation of MOV R4,A(R2).

Reverse Polish Notation (1)



Each railroad car represents one symbol in the formula to be converted from infix to reverse Polish notation.

Reverse Polish Notation (2)

Most recently arrived car on the Texas line	Car at the switch: ⊥	+	−	x	/	(	)
⊥	4	1	1	1	1	1	5
+	2	2	2	1	1	1	2
−	2	2	2	1	1	1	2
x	2	2	2	2	2	1	2
/	2	2	2	2	2	1	2
(	5	1	1	1	1	1	3

Decision table used by the infix-to-reverse Polish notation algorithm

Reverse Polish Notation (3)

Infix	Reverse Polish notation
A + B × C	A B C × +
A × B + C	A B × C +
A × B + C × D	A B × C D × +
(A + B) / (C – D)	A B + C D – /
A × B / C	A B × C /
((A + B) × C + D)/(E + F + G)	A B + C × D + E F + G + /

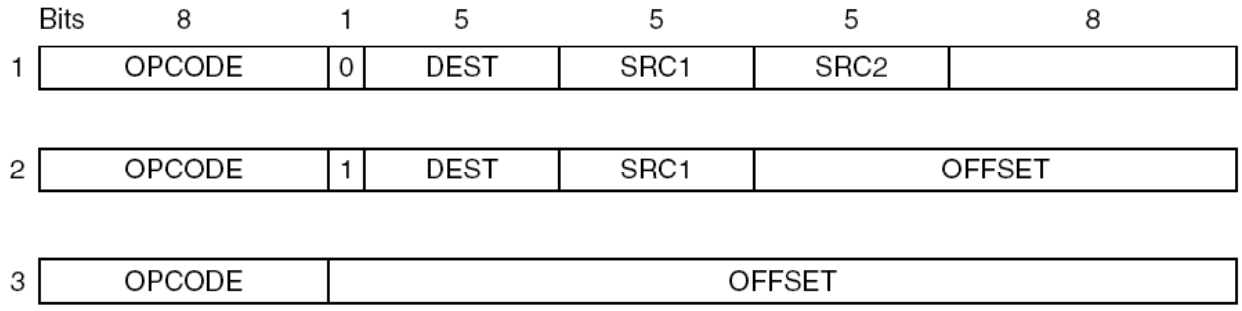
Some examples of infix expressions and their reverse Polish notation equivalents.

Evaluation of Reverse Polish notation Formulas

Step	Remaining string	Instruction	Stack
1	8 2 5 × + 1 3 2 × + 4 – /	BIPUSH 8	8
2	2 5 × + 1 3 2 × + 4 – /	BIPUSH 2	8, 2
3	5 × + 1 3 2 × + 4 – /	BIPUSH 5	8, 2, 5
4	× + 1 3 2 × + 4 – /	IMUL	8, 10
5	+ 1 3 2 × + 4 – /	IADD	18
6	1 3 2 × + 4 – /	BIPUSH 1	18, 1
7	3 2 × + 4 – /	BIPUSH 3	18, 1, 3
8	2 × + 4 – /	BIPUSH 2	18, 1, 3, 2
9	× + 4 – /	IMUL	18, 1, 6
10	+ 4 – /	IADD	18, 7
11	4 – /	BIPUSH 4	18, 7, 4
12	– /	ISUB	18, 3
13	/	IDIV	6

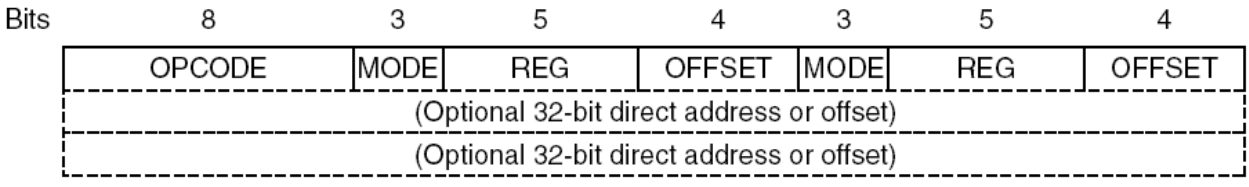
Use of a stack to evaluate a reverse Polish notation formula.

Orthogonality of Opcodes and Addressing Modes (1)



A simple design for the instruction formats of a three-address machine.

Orthogonality of Opcodes and Addressing Modes (2)



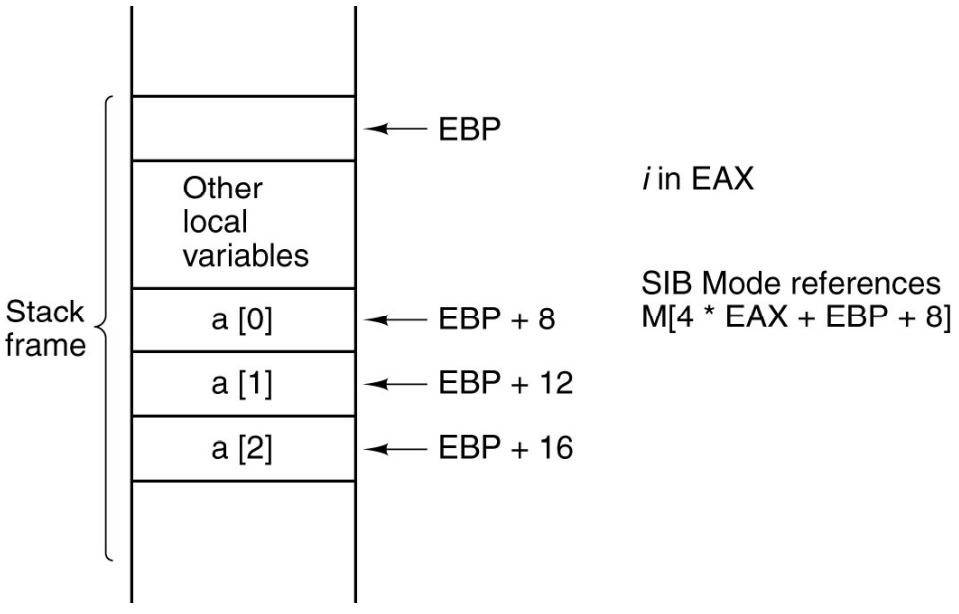
A simple design for the instruction formats of a two-address machine.

The Pentium 4 Addressing Modes (1)

	MOD			
R/M	00	01	10	11
000	M[EAX]	M[EAX + OFFSET8]	M[EAX + OFFSET32]	EAX or AL
001	M[ECX]	M[ECX + OFFSET8]	M[ECX + OFFSET32]	ECX or CL
010	M[EDX]	M[EDX + OFFSET8]	M[EDX + OFFSET32]	EDX or DL
011	M[EBX]	M[EBX + OFFSET8]	M[EBX + OFFSET32]	EBX or BL
100	SIB	SIB with OFFSET8	SIB with OFFSET32	ESP or AH
101	Direct	M[EBP + OFFSET8]	M[EBP + OFFSET32]	EBP or CH
110	M[ESI]	M[ESI + OFFSET8]	M[ESI + OFFSET32]	ESI or DH
111	M[EDI]	M[EDI + OFFSET8]	M[EDI + OFFSET32]	EDI or BH

The Pentium 4 32-bit addressing modes. M[*x*] is the memory word at *x*.

The Pentium 4 Addressing Modes (2)



Access to *a*[*i*].

Discussion of Addressing Modes

Addressing mode	Pentium 4	UltraSPARC III	8051
Accumulator			×
Immediate	×	×	×
Direct	×		×
Register	×	×	×
Register indirect	×	×	×
Indexed	×	×	
Based-indexed		×	
Stack			

A comparison of addressing modes.

Loop Control

```
        i = 1;

L1:     first-statement;
        .
        .
        .
        last-statement;
        i = i + 1;
        if (i < n) goto L1;
```

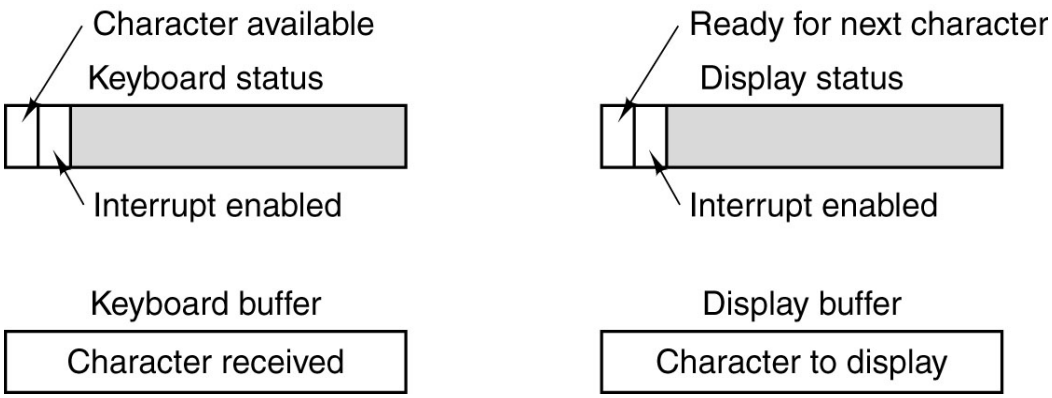
(a)

```
        i = 1;
L1:     if (i > n) goto L2;
        first-statement;
        .
        .
        .
        last-statement
        i = i + 1;
        goto L1;
L2:
```

(b)

(a) Test-at-the-end loop.
(b) Test-at-the-beginning loop.

Input/Output (1)



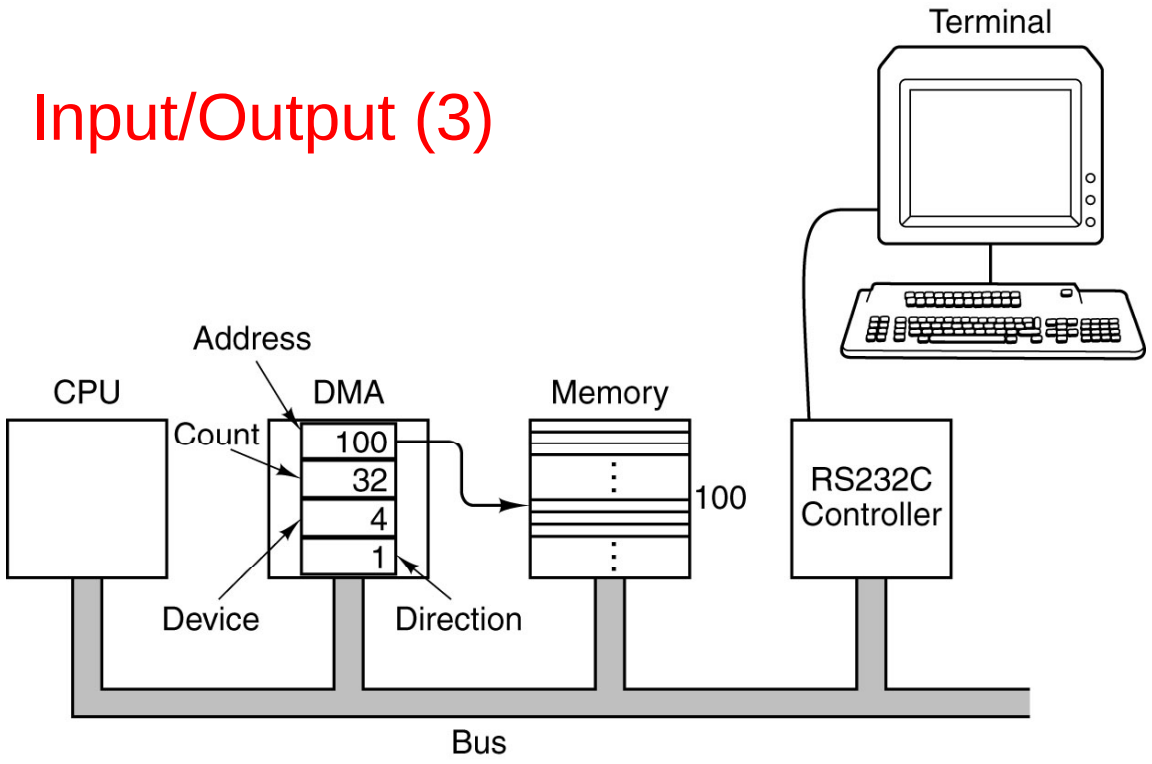
Device registers for a simple terminal.

Input/Output (2)

```
// Output a block of data to the device
int status, i, ready;

for (i = 0; i < count; i++) {
    do {
        status = in(display_status_reg);        // get status
        ready = (status >> 7) & 0x01;           // isolate ready bit
    } while (ready != 1);
    out(display_buffer_reg, buf[i]);
}
}
```

An example of programmed I/O.

Input/Output (3)



A system with a DMA controller.

The Pentium 4 Instructions (1)

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
CMOVcc DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract SRC from DST
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 SRC & carry from DST
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

A selection of the Pentium 4 integer instructions.

The Pentium 4 Instructions (2)

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

A selection of the Pentium 4 integer instructions.

The Pentium 4 Instructions (3)

Test/compare

TEST 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 n	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

A selection of the Pentium 4 integer instructions.

The Pentium 4 Instructions (4)

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
DST = destination

# = shift/rotate count
LV = # locals

A selection of the Pentium 4 integer instructions.

The UltraSPARC III Instructions (1)

Loads

LDSB ADDR,DST	Load signed byte (8 bits)
LDUB ADDR,DST	Load unsigned byte (8 bits)
LDSH ADDR,DST	Load signed halfword (16 bits)
LDUH ADDR,DST	Load unsigned halfword (16)
LDSW ADDR,DST	Load signed word (32 bits)
LDUW ADDR,DST	Load unsigned word (32 bits)
LDX ADDR,DST	Load extended (64-bits)

Stores

STB SRC,ADDR	Store byte (8 bits)
STH SRC,ADDR	Store halfword (16 bits)
STW SRC,ADDR	Store word (32 bits)
STX SRC,ADDR	Store extended (64 bits)

The primary UltraSPARC III integer instructions.

The UltraSPARC III Instructions (2)

Arithmetic

ADD R1,S2,DST	Add
ADDCC "	Add and set icc
ADDC "	Add with carry
ADDCCC "	Add with carry and set icc
SUB R1,S2,DST	Subtract
SUBCC "	Subtract and set icc
SUBC "	Subtract with carry
SUBCCC "	Subtract with carry and set icc
MULX R1,S2,DST	Multiply
SDIVX R1,S2,DST	Signed divide
UDIVX R1,S2,DST	Unsigned divide
TADCC R1,S2,DST	Tagged add

The primary UltraSPARC III integer instructions.

The UltraSPARC III Instructions (3)

Shifts/rotates

SLL R1,S2,DST	Shift left logical (32 bits)
SLLX R1,S2,DST	Shift left logical extended (64)
SRL R1,S2,DST	Shift right logical (32 bits)
SRLX R1,S2,DST	Shift right logical extended (64)
SRA R1,S2,DST	Shift right arithmetic (32 bits)
SRAX R1,S2,DST	Shift right arithmetic ext. (64)

Miscellaneous

SETHI CON,DST	Set bits 10 to 31
MOVcc CC,S2,DST	Move on condition
MOVr R1,S2,DST	Move on register
NOP	No operation
POPC S1,DST	Population count
RDCCR V,DST	Read condition code register
WRCCR R1,S2,V	Write condition code register
RDPC V,DST	Read program counter

The primary UltraSPARC III integer instructions.

The UltraSPARC III Instructions (4)

Boolean

AND R1,S2,DST	Boolean AND
ANDCC "	Boolean AND and set icc
ANDN "	Boolean NAND
ANDNCC "	Boolean NAND and set icc
OR R1,S2,DST	Boolean OR
ORCC "	Boolean OR and set icc
ORN "	Boolean NOR
ORNCC "	Boolean NOR and set icc
XOR R1,S2,DST	Boolean XOR
XORCC "	Boolean XOR and set icc
XNOR "	Boolean EXCLUSIVE NOR
XNORCC "	Boolean EXCL. NOR and set icc

The primary UltraSPARC III integer instructions.

The UltraSPARC III Instructions (5)

Transfer of control

BPcc ADDR	Branch with prediction
BPr SRC,ADDR	Branch on register
CALL ADDR	Call procedure
RETURN ADDR	Return from procedure
JMPL ADDR,DST	Jump and link
SAVE R1,S2,DST	Advance register windows
RESTORE "	Restore register windows
Tcc CC,TRAP#	Trap on condition
PREFETCH FCN	Prefetch data from memory
LDSTUB ADDR,R	Atomic load/store
MEMBAR MASK	Memory barrier

SRC = source register
DST = destination register
R1 = source register
S2 = source: register or immediate
ADDR = memory address

TRAP# = trap number
FCN = function code
MASK = operation type
CON = constant
V = register designator

CC = condition code set
R =destination register
cc = condition
r = LZ,LEZ,Z,NZ,GZ,GEZ

The primary UltraSPARC III integer instructions.

The UltraSPARC III Instructions (6)

Instruction	How to do it
MOV SRC,DST	OR SRC with G0 and store the result DST
CMP SRC1,SRC2	SUBCC SRC2 from SRC1 and store the result in G0
TST SRC	ORCC SRC with G0 and store the result in G0
NOT DST	XNOR DST with G0
NEG DST	SUB DST from G0 and store in DST
INC DST	ADD 1 to DST (immediate operand)
DEC DST	SUB 1 from DST (immediate operand)
CLR DST	OR G0 with G0 and store in DST
NOP	SETHI G0 to 0
RET	JMPL %I7+8,%G0

Some simulated UltraSPARC III instructions.

8051 Instructions (1)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
MOV	Move *src* to ACC		×	×	×	×		
MOV	Move *src* to register	×		×		×		
MOV	Move *src* to memory	×	×	×	×	×		
MOV	Move *src* to indirect RAM	×		×		×		
MOV	Move 16-bit constant to DPTR							
MOVC	Move code to ACC offset from DPTR							
MOVC	Move code to ACC offset from PC							
MOVX	Move external RAM byte to ACC				×			
MOVX	Move ext. RAM byte to ACC @DPTR							
MOVX	Move to ext. RAM byte from ACC				×			
MOVX	Move to ext. RAM byte from ACC @DPTR							
PUSH	Push *src* byte to stack			×				
POP	Pop stack byte to dst			×				
XCH	Exchange ACC and dst	×		×	×			
XCHD	Exchange low-order digit ACC and dst			×				
SWAP	Swap nibbles of dst	×						
ADD	Add *src* to ACC		×	×	×	×		

The 8051 Instruction set.

8051 Instructions (2)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
ADDC	Add *src* to ACC with carry		×	×	×	×		
SUBB	Subtract *src* from ACC with borrow		×	×	×	×		
INC	Increment dst	×	×	×	×			
DEC	Decrement dst	×	×	×	×			
INC	DPTR							
MUL	Multiply							
DIV	Divide							
DA	Decimal adjust dst	×						
ANL	AND *src* to ACC		×	×	×	×		
ANL	AND ACC to dst			×				
ANL	AND immediate to dst			×				
ORL	OR *src* to ACC		×	×	×	×		
ORL	OR ACC to dst			×				
ORL	OR immediate to dst			×				

The 8051 Instruction set.

8051 Instructions (3)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
XRL	XOR *src* to ACC		×	×	×	×		
XRL	XOR ACC to dst			×				
XRL	XOR immediate to dst			×				
CLR	Clear dst	×						
CPL	Complement dst	×						
RL	Rotate dst left	×						
RLC	Rotate dst left through carry	×						
RR	Rotate dst right	×						
RRC	Rotate dst right through carry	×						

The 8051 Instruction set.

8051 Instructions (4)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
CLR	Clear bit						×	×
SETB	Set bit						×	×
CPL	Complement bit						×	×
ANL	AND *src* to carry							×
ANL	AND complement of *src* to carry							×
ORL	OR *src* to carry							×
ORL	OR complement of *src* to carry							×
MOV	Move *src* to carry							×
MOV	Move carry to src							×
JV	Jump relative if carry set							
JNC	Jump relative if carry not set							
JB	Jump relative if direct bit set							×
JNB	Jump relative if direct bit not set							×
JBC	Jump rel. if direct bit set and carry clear							×

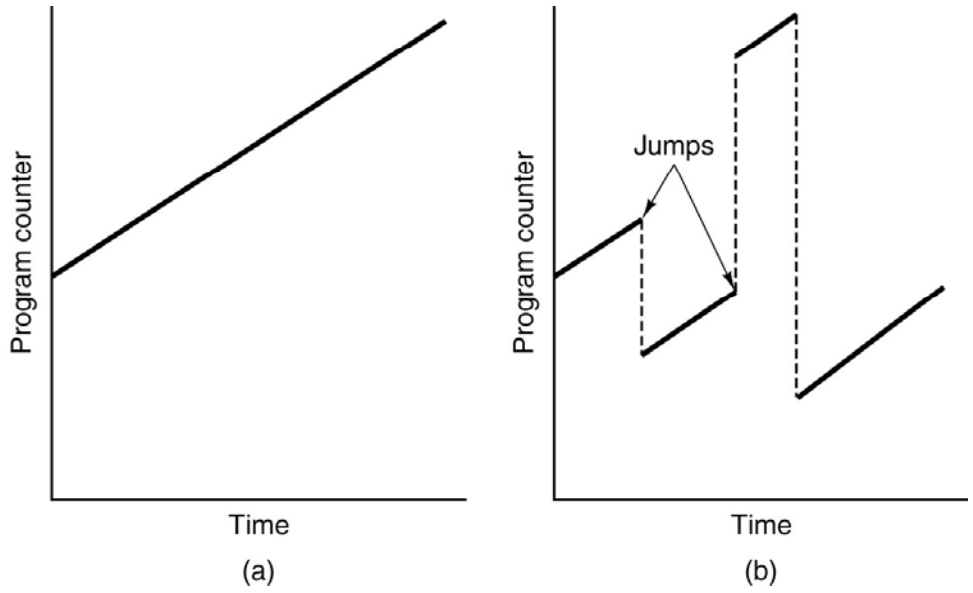
The 8051 Instruction set.

8051 Instructions (5)

Inst.	Description	ACC	Reg	Dir	@R	#	C	Bit
ACALL	Call subroutine (11-bit addr)							
LCALL	Call subroutine (16-bit addr)							
RET	Return from subroutine							
RETI	Return from interrupt							
SJMP	Short relative jump (8-bit addr)							
AJMP	Absolute jump (11-bit addr)							
LJMP	Absolute jump (16-bit addr)							
JMP	Jump indirect rel. to DPR+ACC							
JZ	Jump if ACC is zero							
JNZ	Jump if ACC is nonzero							
CJNE	Comp. *src* to ACC, jump unequal			×		×		
CJNE	Comp. *src* to immediate, jump unequal		×		×			
DJNZ	Decrement dst and jump nonzero							
NOP	No operation							

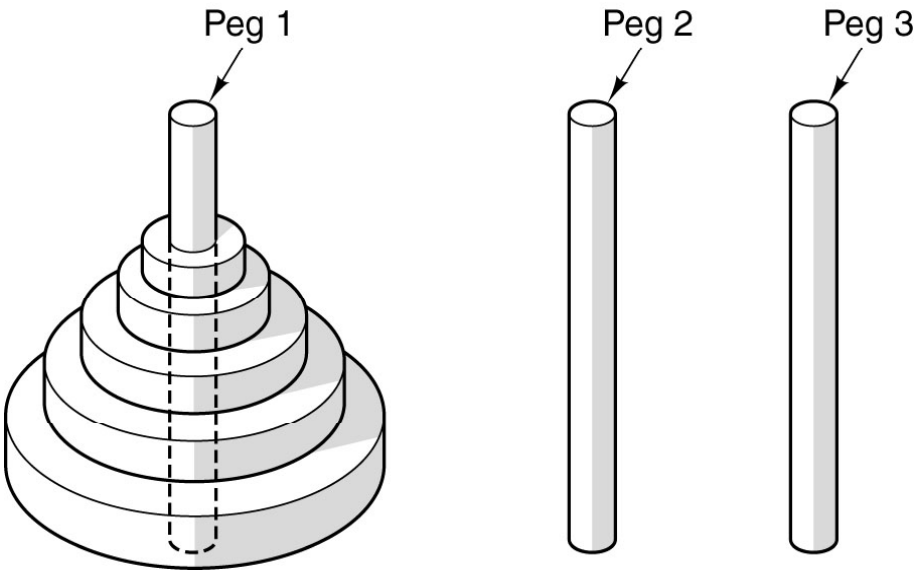
The 8051 Instruction set.

Sequential Flow of Control and Branches



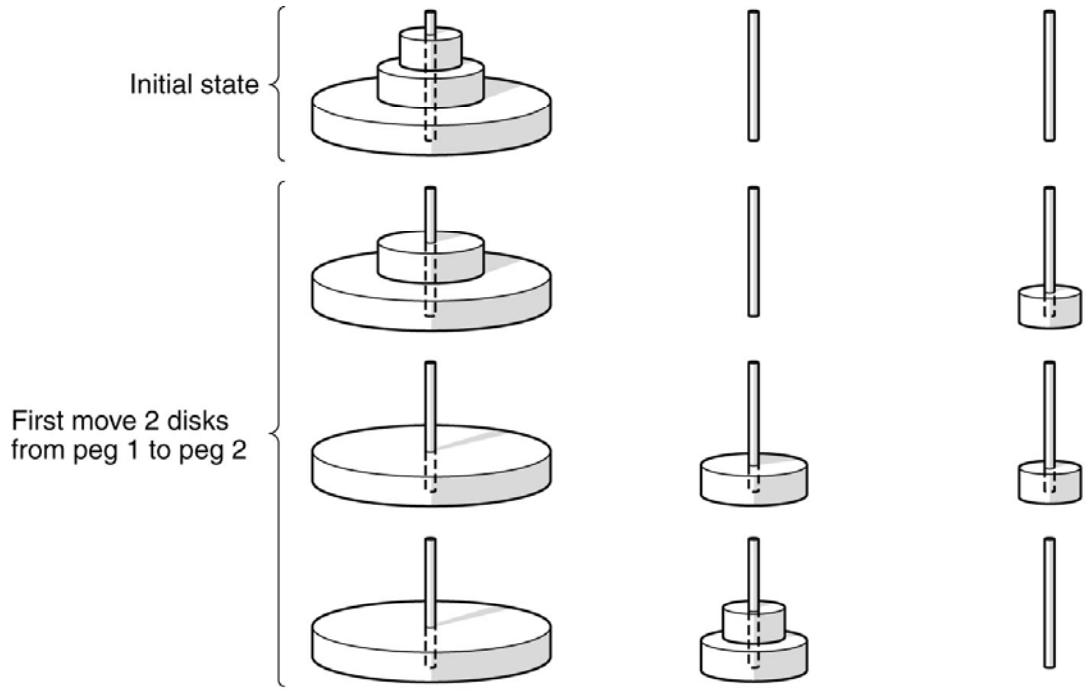
Program counter as a function of time (smoothed).
(a) Without branches. (b) With branches.

Recursive Procedures (1)



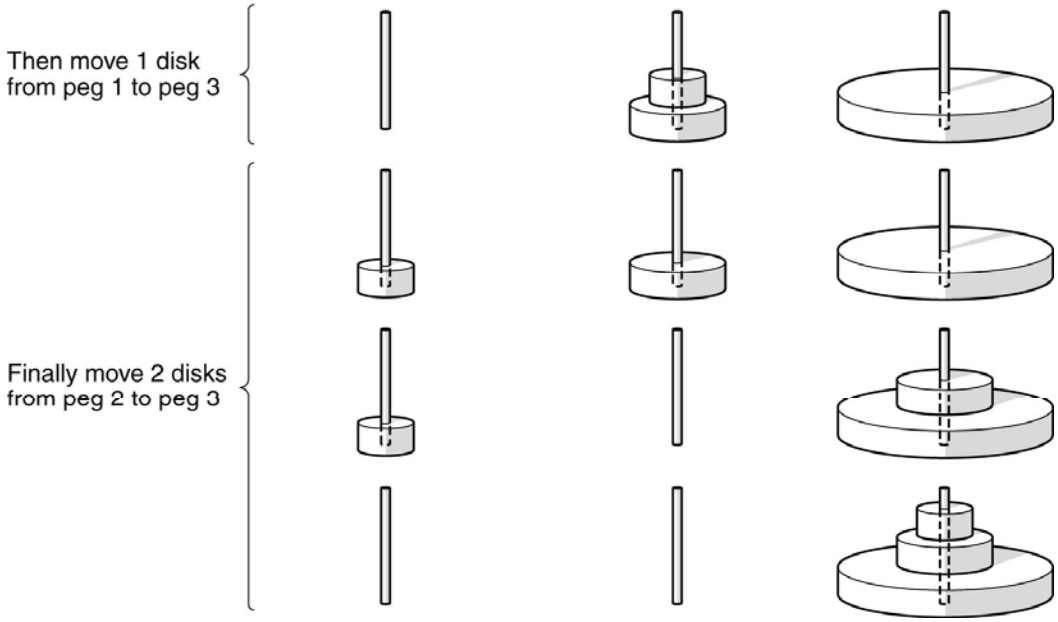
Initial configuration for the Towers of Hanoi problem for five disks.

Recursive Procedures (2)



The steps required to solve the Towers of Hanoi for three disks.

Recursive Procedures (3)



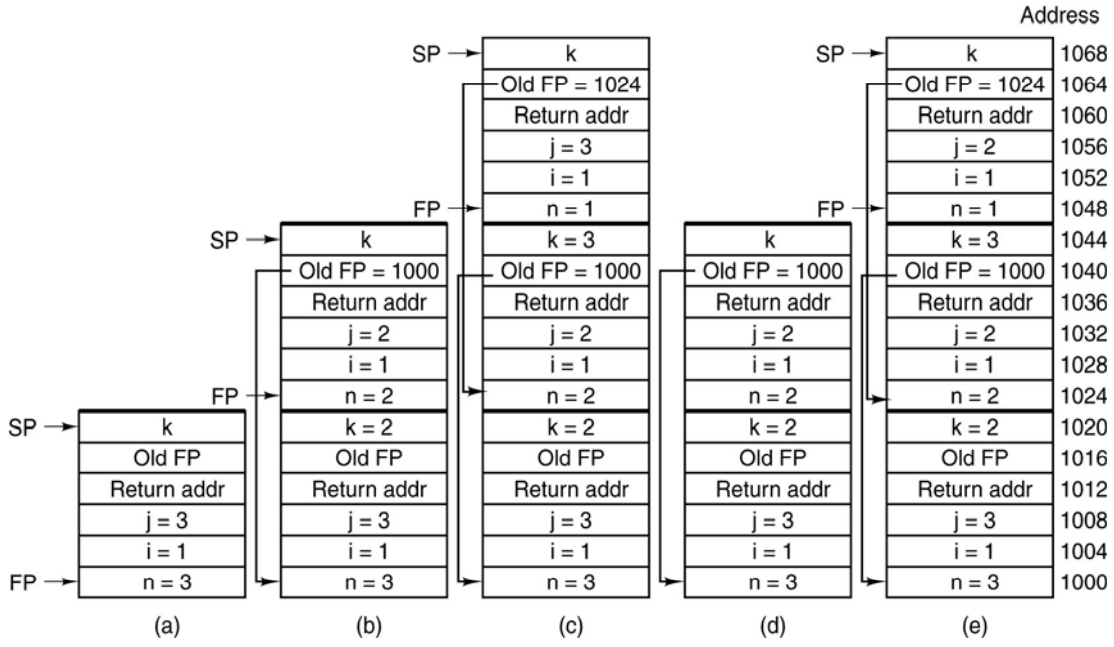
The steps required to solve the Towers of Hanoi for three disks.

Recursive Procedures (4)

```
public void towers(int n, int i, int j) {
   int k;

   if (n == 1)
      System.out.println("Move a disk from " + i + " to " + j);
   else {
      k = 6 – i – j;
      towers(n – 1, i, k);
      towers(1, i, j);
      towers(n – 1, k, j);
   }
}
```

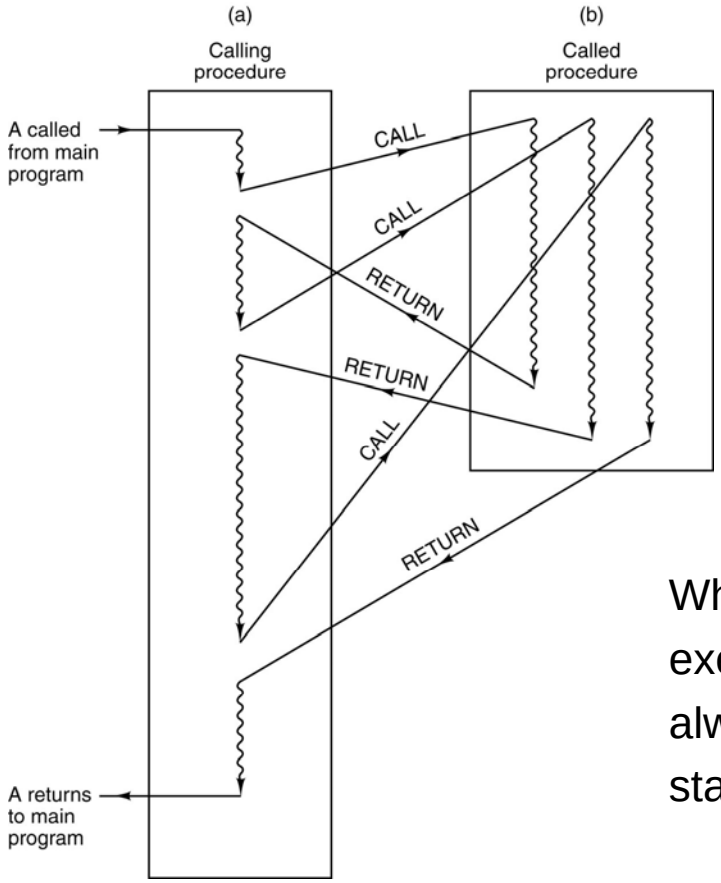
A procedure for solving the Towers of Hanoi.

Recursive Procedures (5)



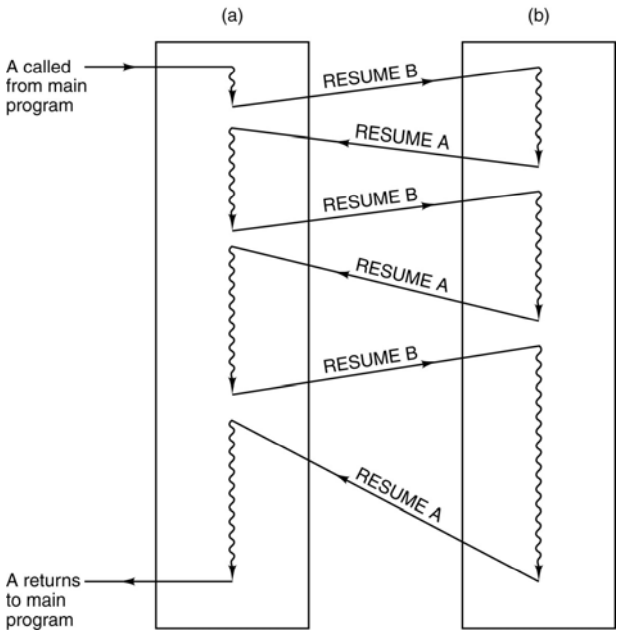
The stack at several points during the execution of Fig. 5-42.

Coroutines (1)



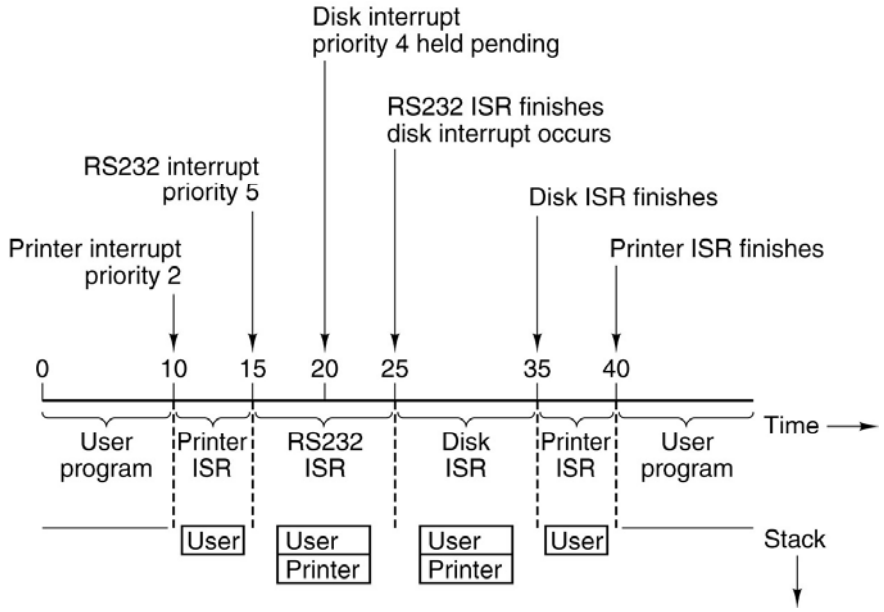
When a procedure is called, execution of the procedure always begins at the first statement of the procedure.

Coroutines (2)



When a coroutine is resumed, execution begins at the statement where it left off the previous time, not at the beginning.

Interrupts



Time sequence of multiple interrupt example.

Towers of Hanoi in Pentium 4 Assembly Language (1)

```
          .586                          ; compile for Pentium (as opposed to 8088 etc.)
.MODEL FLAT
PUBLIC _towers                          ; export 'towers'
EXTERN _printf:NEAR                     ; import printf
.CODE
 _towers:  PUSH EBP                     ; save EBP (frame pointer) and decrement ESP
           MOV EBP, ESP                 ; set new frame pointer above ESP
           CMP [EBP+8], 1               ; if (n == 1)
           JNE L1                       ; branch if n is not 1
           MOV EAX, [EBP+16]            ; printf(" ...", i, j);
           PUSH EAX                     ; note that parameters i, j and the format
           MOV EAX, [EBP+12]            ; string are pushed onto the stack
           PUSH EAX                     ; in reverse order. This is the C calling convention
           PUSH OFFSET FLAT:format      ; offset flat means the address of format
           CALL _printf                 ; call printf
           ADD ESP, 12                  ; remove params from the stack
           JMP Done                     ; we are finished

                  . . .
```

Towers of Hanoi for Pentium 4.

Towers of Hanoi in Pentium 4 Assembly Language (2)

```
                .  .  .
L1:     MOV EAX, 6              ; start k = 6 – i – j
        SUB EAX, [EBP+12]       ; EAX = 6 – i
        SUB EAX, [EBP+16]       ; EAX = 6 – i – j
        MOV [EBP+20], EAX       ; k = EAX
        PUSH EAX                ; start towers(n – 1, i, k)
        MOV EAX, [EBP+12]       ; EAX = i
        PUSH EAX                ; push i
        MOV EAX, [EBP+8]        ; EAX = n
        DEC EAX                 ; EAX = n – 1
        PUSH EAX                ; push n – 1
        CALL _towers            ; call towers(n – 1, i, 6 – i – j)
        ADD ESP, 12             ; remove params from the stack
        MOV EAX, [EBP+16]       ; start towers(1, i, j)
        PUSH EAX                ; push j
        MOV EAX, [EBP+12]       ; EAX = i
        PUSH EAX                ; push i
        PUSH 1                  ; push 1
        CALL _towers            ; call towers(1, i, j)
                .  .  .
```

Towers of Hanoi for Pentium 4.

Towers of Hanoi in Pentium 4 Assembly Language (3)

```
                .  .  .
        ADD ESP, 12             ; remove params from the stack
        MOV EAX, [EBP+12]       ; start towers(n – 1, 6 – i – j, i)
        PUSH EAX                ; push i
        MOV EAX, [EBP+20]       ; EAX = k
        PUSH EAX                ; push k
        MOV EAX, [EBP+8]        ; EAX = n
        DEC EAX                 ; EAX = n–1
        PUSH EAX                ; push n – 1
        CALL _towers            ; call towers(n – 1, 6 – i – j, i)
        ADD ESP, 12             ; adjust stack pointer
Done:   LEAVE                   ; prepare to exit
        RET 0                   ; return to the caller

.DATA
format  DB "Move disk from %d to %d\n"  ; format string
END
```

Towers of Hanoi for Pentium 4.

Towers of Hanoi in UltraSPARC III Assembly Language (1)

```
#define N %i0                          /* N is input parameter 0 */
#define I %i1                          /* I is input parameter 1 */
#define J %i2                          /* J is input parameter 2 */
#define K %l0                          /* K is local variable 0 */
#define Param0 %o0                     /* Param0 is output parameter 0 */
#define Param1 %o1                     /* Param1 is output parameter 1 */
#define Param2 %o2                     /* Param2 is output parameter 2 */
#define Scratch %l1                    /* as an aside, cpp uses the C comment convention */
        .proc 04
        .global towers

towers: save %sp, –112, %sp
        cmp N, 1                       ! if (n == 1)
        bne Else                       ! if (n != 1) goto Else

        sethi %hi(format), Param0      ! printf("Move a disk from %d to %d\n", i, j)
        or Param0, %lo(format), Param0 ! Param0 = address of format string
        mov I, Param1                  ! Param1 = i
        call printf                    ! call printf BEFORE parameter 2 (j) is set up
        mov J, Param2                  ! use the delay slot after call to set up parameter 2
        b Done                         ! we are done now
        nop                            ! fill delay slot
```

• • •

Towers of Hanoi for UltraSPARC III.

Towers of Hanoi in UltraSPARC III Assembly Language (2)

• • •

```
Else:   mov 6, K                       ! start k = 6 –i – j
        sub K, J, K                    ! k = 6 – j
        sub K, I, K                    ! k = 6 – i – j

        add N, –1, Scratch             ! start towers(n – 1, i, k)
        mov Scratch, Param0            ! Scratch = n – 1
        mov I, Param1                  ! parameter 1 = i
        call towers                    ! call towers BEFORE parameter 2 (k) is set up
        mov K, Param2                  ! use the delay slot after call to set up parameter 2

        mov 1, Param0                  ! start towers(1, i, j)
        mov I, Param1                  ! parameter 1 = i
        call towers                    ! call towers BEFORE parameter 2 (j) is set up
        mov J, Param2                  ! parameter 2 = j
```

• • •

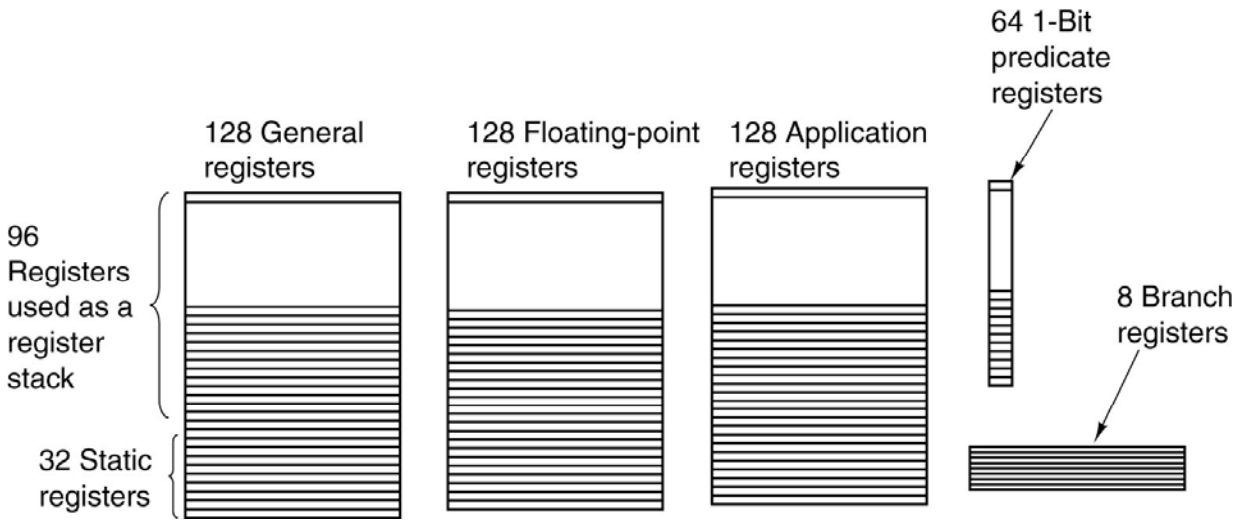
Towers of Hanoi for UltraSPARC III.

Towers of Hanoi in UltraSPARC III Assembly Language (3)

```
. . .

        mov Scratch, Param0          ! start towers(n – 1, k, j)
        mov K, Param1                ! parameter 1 = k
        call towers                  ! call towers BEFORE parameter 2 (j) is set up
        mov J, Param2                ! parameter 2 = j

Done:   ret                          ! return
        restore                      ! use the delay slot after ret to restore windows

format: .asciz "Move a disk from %d to %d\n"
```

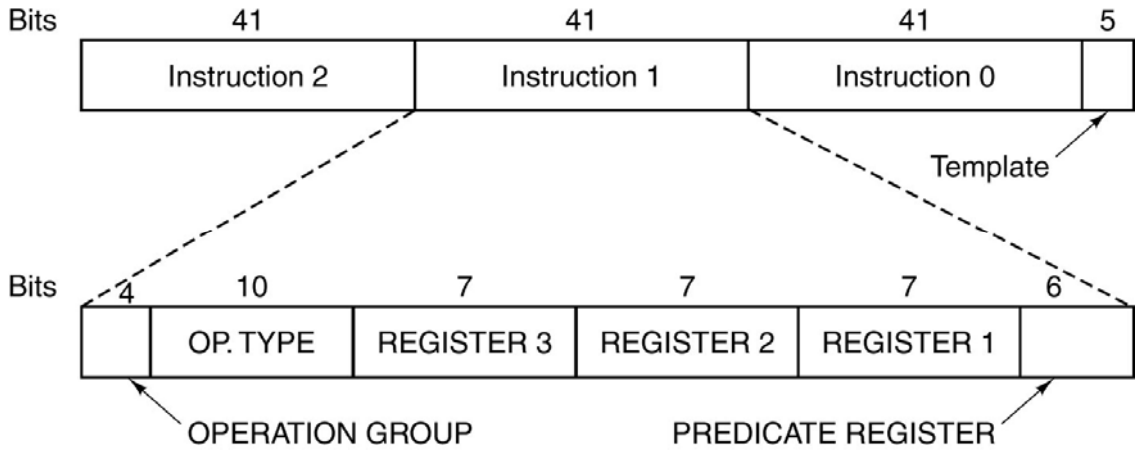
Towers of Hanoi for UltraSPARC III.

Reducing Memory References



The Itanium 2's registers.

Instruction Scheduling



An IA-64 bundle contains three instructions.

Reducing Conditional Branches: Predication (1)

```
if (R1 == 0)
    R2 = R3;
```

(a)

```
    CMP R1,0
    BNE L1
    MOV R2,R3
L1:
```

(b)

```
CMOVZ R2,R3,R1
```

(c)

(a) An if statement.
(b) Generic assembly code for a).
(c) A conditional instruction.

Reducing Conditional Branches: Predication (2)

```
if (R1 == 0) {
    R2 = R3;
    R4 = R5;
} else {
    R6 = R7;
    R8 = R9;
}
```

(a)

```
    CMP R1,0
    BNE L1
    MOV R2,R3
    MOV R4.R5
    BR L2
L1: MOV R6,R7
    MOV R8,R9
L2:
```

(b)

```
CMOVZ R2,R3,R1
CMOVZ R4,R5,R1
CMOVN R6,R7,R1
CMOVN R8,R9,R1
```

(c)

(a) An if statement.
(b) Generic assembly code for a).
(c) Conditional instruction.

Reducing Conditional Branches: Predication (3)

```
if (R1 == R2)
    R3 = R4 + R5;
else
    R6 = R4 – R5
```

(a)

```
    CMP R1,R2
    BNE L1
    MOV R3,R4
    ADD R3,R5
    BR L2
L1: MOV R6,R4
    SUB R6,R5
L2:
```

(b)

```
CMPEQ R1,R2,P4
<P4> ADD R3,R4,R5
<P5> SUB R6,R4,R5
```

(c)

(a) An if statement.
(b) Generic assembly code for a).
(c) Predicated instruction.