Computer Structure and Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

Registers

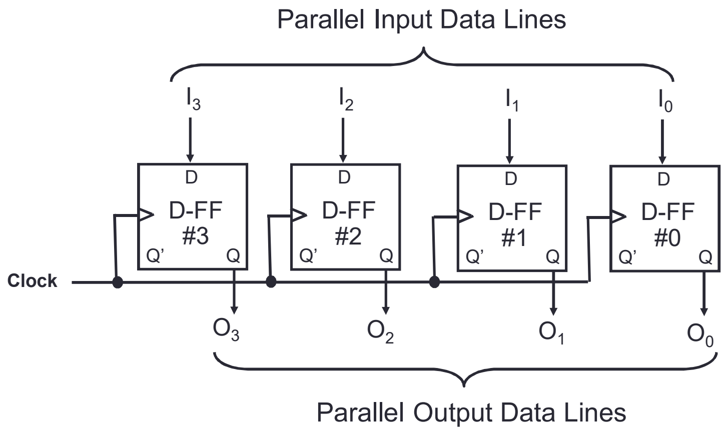
An n-bit Register is made of n flip-flops controlled in group.

Based on the availability of flip-flops' output and their access method, registers can be:

- **PIPO**: Parallel In, Parallel Out
- **SIPO**: Serial In, Parallel Out
- **PISO**: Parallel In, Serial Out
- **SISO**: Serial In, Serial Out

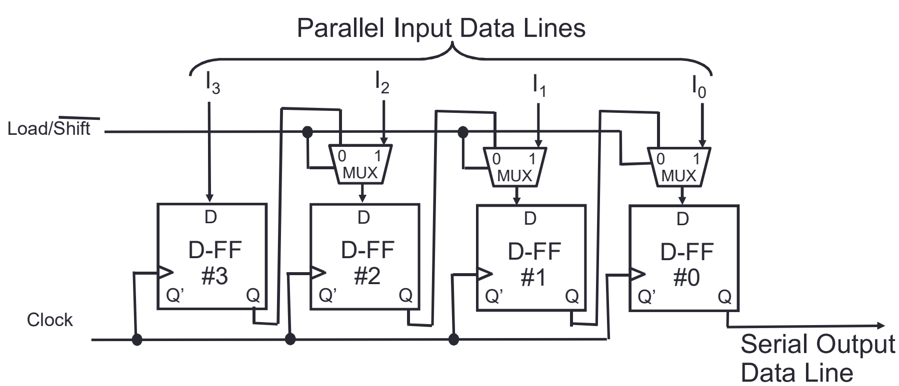
Registers

Example: The logic diagram of a 4-bit PIPO register.



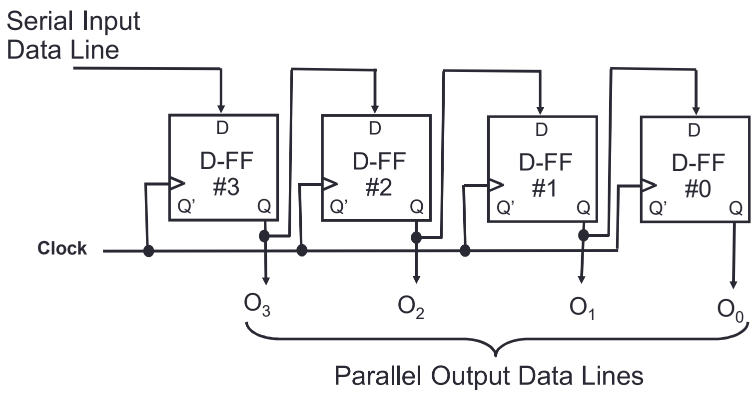
Registers

Example: The logic diagram of a 4-bit PISO register.



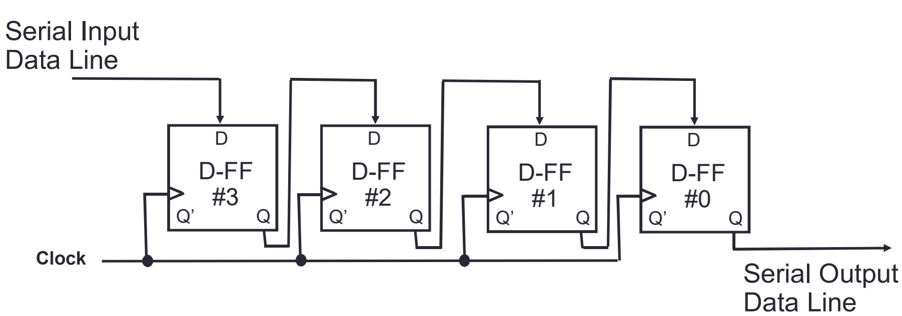
Registers

Example: The logic diagram of a 4-bit SIPO register.



Registers

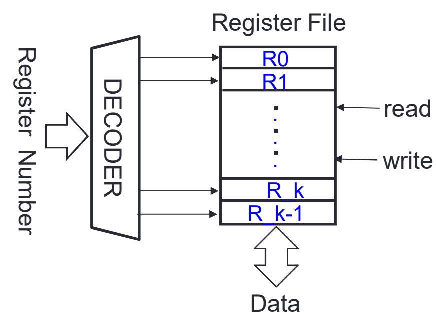
Example: The logic diagram of a 4-bit SISO register.



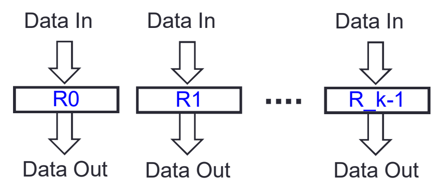
Registers

From the implementation point of view, registers can be:

- **Integrated** (Register File): It is like a small memory where each location refer to one register. All registers are accessed through internal address and data buses.
→ Only one register can be accessed at a given time.



- **Isolated**: Registers can be accessed in parallel as their access lines (input and output lines) are separate.



Registers

From the addressability point of view, registers can be:

- **Addressable**: Assembly programmer can address them in the instructions.
e.g. R0, R1, …, R15 in IBM360 family
or AX, BX, CX, DX, … in 8086/88 processor
We will see them later when learning their assembly programming.

- **Non-addressable**: They are not addressable by the programmer, but are used in the machine.
e.g. MAR, MBR, PC, IR, …
We will see them later.

Registers

From the usage point of view, registers can be:

- **General purpose registers**: Addressable registers that can be used for all purposes defined by the machine instruction set.
  e.g. Registers 0, 1, …15 in IBM360/370 family.

- **Special purpose registers:** Addressable registers that may be used for special purposes defined by some machine instruction set.
  e.g. Registers CX (for counting) or DX (for division, multiplication, and I/O instructions in 8086/88 family.

Internal Machine Registers

Machine non-addressable registers include:

- **MAR** (Memory Address Register) keeps memory location address during a memory cycle.
- **MBR** (Memory Buffer Register) holds the data to be read/written from/into memory.
- **IR** (Instruction Register) holds the instruction fetched from main memory into CU.
- **PC** (Program Counter) holds the address of the next instruction to be fetched from main memory and executed.

We will see some other registers later.

Machine addressable registers include general-purpose and special-purpose registers.

The number of such registers and their purpose are different from machine to machine.

Addressing modes

1. **Direct addressing**: Address of operand is given in the IF (instruction format). It can be:
   - Register direct (register number is given in IF)
   - Memory direct (memory address is given in IF)

Example 1: mov ax, array ; in 8086/88 processor

Here, ax is a symbolic address for register ax which is included in the IF. Array is also a symbolic address mentioned in the IF to directly address a word in main memory.

Example 2: L 2,NUM ; in IBM 360 family

Here, 2 indicates register#2 and NUM is the symbolic address of a word in main memory.

Almost all machines employ direct addressing modes.

Addressing modes

2. **Indirect addressing**: The address included in the IF indicates the register or memory location that contains the address of the operand in memory). It can be:
   - Register indirect
   - Memory indirect

Example 1: L 2,0(3) ; in IBM360 family

Here, register 2 (indicated in the IF) is loaded with a word in main memory whose address is in register 3.

Example 2: call far ptr [p1] ; in 8086/88 processor

A procedure is called whose address is stored in a word stored in location p1. p1 is the symbolic address of the pointer included in the IF, i.e. (M_{p1}) = address of procedure.

Almost all machines employ indirect addressing modes.

Addressing modes

2. **Indirect addressing**:
Indirect addressing can be of 1, 2, or more levels.

The 1-level indirect addressing uses one indirection, like we saw in the previous slide.
→ $M_{(M\_addr)}$ is the actual operand address.

The 2-level indirect addressing use two consecutive indirection to get the actual operand.
→ $M_{(M\_(M\_addr))}$ is the actual operand address.

… and so on.

Most machines with indirect addressing use one level indirection; very few use two levels of indirection.

Example Machine 3:

In a 2-address machine, we have:

- Main memory size: 2^{16} addressable units (each 8 bits)
- Word size: 16 bits, unaligned, big endian
- Addressing modes: Memory Direct, Memory Indirect
- Conditional jump instructions to make loops
- The instruction format shown below:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Example Machine 3:

Instructions include:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Opcode	Mnemonic		Operation
00000000	mov	addr1,addr2	$M_{addr1} \leftarrow (M_{addr2})$;
00000001	mov	addr1,(addr2)	$M_{addr1} \leftarrow (M\_(M_{addr2}))$;
00000010	mov	(addr1),addr2	$M\_(M_{addr1}) \leftarrow (M_{addr2})$;
00000011	add	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) + (M_{addr2})$;
00000100	sub	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) - (M_{addr2})$;
00000101	add	addr1,(addr2)	$M_{addr1} \leftarrow (M_{addr1}) + (M\_(M_{addr2}))$;
00000110	sub	addr1,(addr2)	$M_{addr1} \leftarrow (M_{addr1}) - (M\_(M_{addr2}))$;
10000000	jnz	addr1,addr2	if $(M_{addr1}) \neq 0$ then PC $\leftarrow$ addr2;
10000001	jz	addr1,addr2	if $(M_{addr1}) = 0$ then PC $\leftarrow$ addr2;
10000010	jneg	addr1,addr2	if $(M_{addr1}) < 0$ then PC $\leftarrow$ addr2;
10000011	jpos	addr1,addr2	if $(M_{addr1}) > 0$ then PC $\leftarrow$ addr2;

Example Machine 3:

- What is the size of machine registers?
  L_{MAR} = 16 bits; L_{MBR} = 16 bits;
  L_{IR} = 40 bits; L_{PC} = 16 bits;
- Write an assembly program to calculate the sum of first 100 elements of alternating Fibonacci sequence as:
  0, -1, 1, -2, 3, -5, 8, -13, 21, -34, …
- Translate your assembly program to machine language.
- Write a program to add the elements of an array of 100 words.
- Translate your assembly program to machine language.

Example Machine 3:

Write an assembly program to sum up the first 100 elements of series 0, -1, 1, -2, 3, -5, 8, -13, 21, -34, ….

```
        org     0
loop:
        add     sum,curr      ; add current element to sum
        sub     sum,next      ; subtract next element from sum
        add     curr,next     ; create the next element in curr
        add     next,curr     ; create another next element in next
        sub     count,one     ; reduce count by 1
        jnz     count,loop    ; if not zero go to loop

curr:   dw      0             ; variable to keep current element
next:   dw      1             ; variable to keep next element
sum:    dw      0             ; variable to keep sum of elements
count:  dw      50            ; variable to count 50 times
one:    dw      1             ; constant 1
        end                   ; end of program
```


Example Machine 3:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Translation to machine code:

```
Address  Machine Code  Assembly Program
                               org    0h
0000                   loop:
0000     030022001E            add    sum,curr     ; add current eleme
0005     0400220020            sub    sum,next     ; subtract next elem
000A     03001E0020            add    curr,next    ; create the next ele
000F     030020001E            add    next,curr    ; create another nex
0014     0400240026            sub    count,one    ; reduce count by 1
0019     8000240000            jnz    count,loop   ; if not zero go to lo

001E     0000          curr:   dw     0            ; variable to keep cu
0020     0001          next:   dw     1            ; variable to keep ne
0022     0000          sum:    dw     0            ; variable to keep su
0024     0032          count:  dw     50           ; variable to count 5
0026     0001          one:    dw     1            ; constant 1
                               end                 ; end of program
```

Example Machine 3:

Write an assembly program to add the elements of an array of 100 words.

```
        org     100h            ; start address of the program @ 100h
loop:
        add     sum,(arrptr)    ; add the array element pointed by (arrptr)
        add     arrptr,two      ; calculate the pointer for the next element
        sub     count,two       ; reduce count by 2
        jnz     count,loop      ; if not zero go to loop

arrptr: dw      array           ; variable to point to array elements
sum:    dw      0               ; variable to keep sum of elements
count:  dw      ٢00             ; variable to count 100 times
two:    dw      2               ; constant 2
array:  dw      ١٠0 dup(?)      ; array of ١٠0 word elements
        end                     ; end of program
```

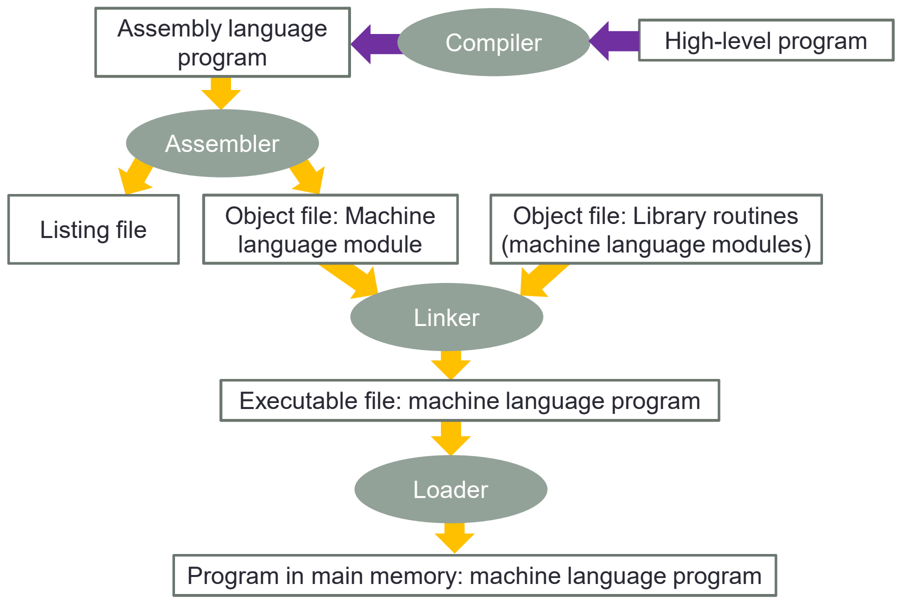

Example Machine 3:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Translation to machine code:

Address	Machine Code	Assembly Program			
			org	100h	; start address of th
0100		loop:			
0100	0501160114		add	sum,(arrptr)	; add the array elen
0105	030114011A		add	arrptr,two	; calculate the poin
010A	040118011A		sub	count,two	; reduce count by 2
010F	8001180100		jnz	count,loop	; if not zero go to lo
0114	011C	arrptr:	dw	array	; variable to point t
0116	0000	sum:	dw	0	; variable to keep s
0118	00C8	count:	dw	200	; variable to count
011A	0002	two:	dw	2	; constant 2
011C	???? ???? ????	array:	dw	100 dup(?)	; array of 100 word
			end		; end of program

Assembly program execution flow:



Assembler:

Assembler converts assembly language into object files. It also creates listing files that contains readable text.

Object files contains a combination of machine instructions, data, and information needed to place program instructions and data properly in memory.

Most Assemblers work in two passes:

Pass 1: Reads each line of the source program and records all labels in a Symbol Table.

Pass 2: Uses information in the Symbol Table to produce the actual machine code for each line of source program.

Object file format:

Object file header	Text segment	Data segment	Relocation information	Symbol table	Debugging information

- Object file header describes the size and position of the other pieces of the file
- Text segment contains the machine instructions
- Data segment contains binary representation of data in assembly file
- Relocation info identifies instructions and data that depend on absolute addresses
- Symbol table associates addresses with external labels and lists unresolved references
- Debugging info

Linker:

A tool that merges the object files produced by separate compilation/assembly and creates an executable file.

It has three tasks:

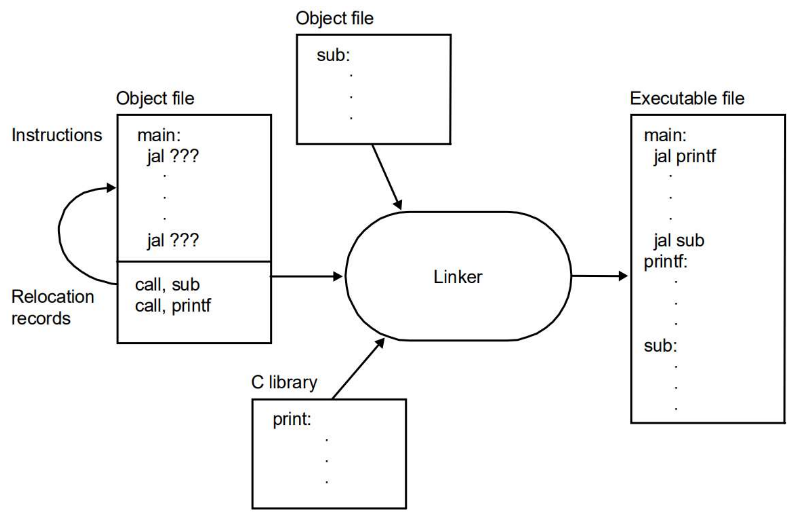
1. Searches the program to find library routines used by program, e.g. printf(), scanf(), math routines, ….

2. Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references

3. Resolve references among files

Linker:

Tool that merges the object files produced by separate compilation/assembly and creates an executable file.



Loader:

A part of Operating System that brings an executable file residing on disk into memory and starts its running.

It is done through the following steps:

1. Read executable file's header to determine the size of text and data segments;
2. Creates (via OS) a new address space for the program;
3. Copies instruction and data into allocated address space;
4. Copies arguments passed to the program on the stack;
5. Initialize the machine registers including stack pointer;
6. Jumps to a startup routine that copies the program's arguments from the stack to registers and calls the program's main routine by loading the PC.

Example Machine 3:

In a 2-address machine, we have:

- Main memory size: 2^{16} addressable units (each 8 bits)
- Word size: 16 bits, unaligned, big endian
- Addressing modes: Memory Direct, Memory Indirect
- Conditional jump instructions to make loops
- The instruction format shown below:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Example Machine 3:

Instructions include:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Opcode	Mnemonic		Operation
00000000	mov	addr1,addr2	$M_{addr1} \leftarrow (M_{addr2})$;
00000001	mov	addr1,(addr2)	$M_{addr1} \leftarrow (M\_(M_{addr2}))$;
00000010	mov	(addr1),addr2	$M\_(M_{addr1}) \leftarrow (M_{addr2})$;
00000011	add	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) + (M_{addr2})$;
00000100	sub	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) - (M_{addr2})$;
00000101	add	addr1,(addr2)	$M_{addr1} \leftarrow (M_{addr1}) + (M\_(M_{addr2}))$;
00000110	sub	addr1,(addr2)	$M_{addr1} \leftarrow (M_{addr1}) - (M\_(M_{addr2}))$;
10000000	jnz	addr1,addr2	if $(M_{addr1}) \neq 0$ then PC $\leftarrow$ addr2;
10000001	jz	addr1,addr2	if $(M_{addr1})=0$ then PC $\leftarrow$ addr2;
10000010	jneg	addr1,addr2	if $(M_{addr1})<0$ then PC $\leftarrow$ addr2;
10000011	jpos	addr1,addr2	if $(M_{addr1})>0$ then PC $\leftarrow$ addr2;

Example Machine 3:

Write an assembly program to add the elements of an array of ١٠0 words.

```
0100       org     100h          ; start a
      loop:
           add     sum,(arrptr)  ; add th
           add     arrptr,two    ; calcul
           sub     count,two     ; reduc
           jnz     count,loop    ; if not zero go to loop

      arrptr: dw   array   011C  ; variab
      sum:    dw   0       0000  ; variab
      count:  dw   ٢00     00C8  ; variab
      two:    dw   2       0002  ; consta
011C  array:  dw   ١٠0 dup(?)????  ; array
           end             ????  ; end of program
                           .......
```

What happens after the last loop iteration?

011C 0000 00C8 0002
???? ???? ???? ????

We must return control to OS at the end.

Example Machine 3:

Suppose we do not have Indirect addressing mode!

Instructions include:

Opcode	addr1	addr2
8 bits	16 bits	16 bits

Now, write an assembly program to add the elements of an array of 100 words.

Opcode	Mnemonic		
00000000	mov	addr1,addr2	
~~00000001~~	~~mov~~	~~addr1,(addr2)~~	
~~00000010~~	~~mov~~	~~(addr1),addr2~~	$M\_(M_{addr1}) \leftarrow (M_{addr2})$;
00000011	add	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) + (M_{addr2})$;
00000100	sub	addr1,addr2	$M_{addr1} \leftarrow (M_{addr1}) - (M_{addr2})$;
~~00000101~~	~~add~~	~~addr1,(addr2)~~	$\cancel{M_{addr1} \leftarrow (M_{addr1}) + (M\_(M_{addr2}))}$;
~~00000110~~	~~sub~~	~~addr1,(addr2)~~	$\cancel{M_{addr1} \leftarrow (M_{addr1}) - (M\_(M_{addr2}))}$;
10000000	jnz	addr1,addr2	if $(M_{addr1}) \neq 0$ then PC $\leftarrow$ addr2;
10000001	jz	addr1,addr2	if $(M_{addr1}) = 0$ then PC $\leftarrow$ addr2;
10000010	jneg	addr1,addr2	if $(M_{addr1}) < 0$ then PC $\leftarrow$ addr2;
10000011	jpos	addr1,addr2	if $(M_{addr1}) > 0$ then PC $\leftarrow$ addr2;

Example Machine 3:

Suppose we do not have Indirect addressing mode!

Write an assembly program to add the elements of an array of ١٠0 words.

Opcode	addr1	addr2
8 bits	16 bits	16 bits

```
        org     100h
loop:
        add     sum,array
        add     loop+3,two
        sub     count,two
        jnz     count,loop

sum:    dw      0
count:  dw      ٢00
two:    dw      2
array:  dw      ١٠0 dup(?)
        end
```

Code alternation is useful for encryption or software lock design.
But it is bad from Software Engineering point of view.
It can also make wrong results if not handled carefully when a piece of code has to be executed more than once.

Addressing modes

1. **Direct**
2. **Indirect**
3. **Immediate**: The operand is included in the IF (no register/memory access is required).

Example 1: sub cx,33 ; in 8086/88 processor

Here, 33 is an immediate data which is reduced from the content of cx register.

Example 2: addi $s0,255 ; in MIPS processor

Here, 255 is added to register $s0 as an immediate data.

All available machines employ immediate addressing modes.

Example Machine 4:

In a 2-address machine, we have:

- Main memory size = 2^{12} addressable units (each 4 bits).
- Word size = 16 bits, unaligned, big endian
- 16 general-purpose registers R0, R1, …, R15
- Addressing modes: Memory Direct, Memory Indirect, Register Direct, Register Indirect, Immediate
- 3 different instruction formats

Format I:

Opcode	r1	r2
4 bits	4 bits	4 bits

Format II:

Opcode	r	addr
4 bits	4 bits	12 bits

Format III:

Opcode	r	data
4 bits	4 bits	16 bits

Example Machine 4:

Instructions in each format include:

Format I:

Opcode	r1	r2
4 bits	4 bits	4 bits

Opcode	Mnemonic	Operation
0000	mov r1,r2	r1 ← (r2);
0001	mov r1,(r2)	r1 ← ($M_{(r2)}$);
0010	mov (r1),r2	$M_{(r1)}$ ← (r2);
0011	add r1,r2	r1 ← (r1)+(r2);
0100	sub r1,r2	r1 ← (r1)-(r2);

Example Machine 4:

Instructions in each format include:

Format II:

Opcode	r	addr
4 bits	4 bits	12 bits

Opcode	Mnemonic	Operation
0101	mov r,addr	r ← (M_{addr});
0110	mov addr,r	M_{addr} ← (r);
0111	mov r,(addr)	r ← ($M_{(M\_addr)}$);
1000	mov (addr),r	$M_{(M\_addr)}$ ← (r);
1001	jz r,addr	if (r)=0 then PC ← addr;
1010	jnz r,addr	if (r)≠0 then PC ← addr;
1011	jneg r,addr	if (r)<0 then PC ← addr;
1100	jpos r,addr	if (r)>0 then PC ← addr;

Example Machine 4:

Instructions in each format include:

Format III:

Opcode	r	data
4 bits	4 bits	16 bits

Opcode	Mnemonic	Operation
1101	mov r,#data	r ← data;
1110	add r,#data	r ← (r) + data;
1111	sub r,#data	r ← (r) - data;

Example Machine 4:

- What are the lengths of machine registers?

MAR = ?	MBR = ?	IR = ?	PC = ?	R0…R15 = ?
12 bits	16 bits	24 bits	12 bits	16 bits

- Write a program to add two 10-element arrays A and B and save the resulted vector in 10-element array C.
- Translate your program to machine code.

Example Machine 4:

Write a program to implement vector add C ← A + B;

```
        org     0
        mov     R0,#A       ; R0 points to array A
        mov     R1,#B       ; R1 points to array B
        mov     R2,#C       ; R2 points to array C
        mov     R5,#10      ; R5 is used as loop counter
loop:   mov     R10,(R0)    ; move one element of A into R10
        mov     R11,(R1)    ; move one element of B into R11
        add     R10,R11     ; add them
        mov     (R2),R10    ; store in the corresponding element of C
        add     R0,#4       ; update pointer to next element of A
        add     R1,#4       ; update pointer to next element of B
        add     R2,#4       ; update pointer to next element of C
        sub     R5,#1       ; decrement counter
        jnz     R5,loop     ; if not zero go to loop

A:      dw      10 dup(?)
B:      dw      10 dup(?)
C:      dw      10 dup(?)
        end
```

Example Machine 4:

Opcode	r1	r2
4 bits	4 bits	4 bits

Opcode	r	addr
4 bits	4 bits	12 bits

Opcode	r	data
4 bits	4 bits	16 bits

Translation to machine code:

Address	Machine Code	Assembly Program			
			org	0	
000	D0004۱		mov	R0,#A	; R0 points to array A
006	D1006۹		mov	R1,#B	; R1 points to array B
00C	D2009۱		mov	R2,#C	; R2 points to array C
012	D5000A		mov	R5,#10	; R5 is used as loop counter
018	1A0	loop:	mov	R10,(R0)	; move one element of A into
01B	1B1		mov	R11,(R1)	; move one element of B into
01E	۳AB		add	R10,R11	; add them
021	22A		mov	(R2),R10	; store in the corresponding
024	E00004		add	R0,#4	; update pointer to next elem
02A	E10004		add	R1,#4	
030	E20004		add	R2,#4	
036	F50001		sub	R5,#1	
03C	A5018		jnz	R5,loop	
04۱	???? ???? ????	A:	dw	10 dup(?)	
06۹	???? ???? ????	B:	dw	10 dup(?)	
09۱	???? ???? ????	C:	dw	10 dup(?)	
			end		

Symbol Table

Symbol	Address
loop	018h
A	04۱h
B	06۹h
C	09۱h

Addressing modes

1. **Direct**
2. **Indirect**
3. **Immediate**
4. **Indexed**: Operand's address is formed by adding the content of an index register and a given address (both index register and the indexed address appear in the IF).

Example: sub ax,array[bx] ; in 8086/88 processor

Here, array is a symbolic address which is added to (bx) to form the memory address location.

It does: ax ← (ax) – ($M_{array+(bx)}$);

Most available machines employ indexed addressing modes.

Example Machine 5:

In a 2/3 address machine, we have:

- Main memory size = 2^{16} addressable units (each 8 bits).
- Word size = 16 bits, unaligned, big endian
- 64 general-purpose registers R0, R1, …, R63
- Addressing modes: Register Direct, Memory Direct, Immediate, Indexed
- 3 instruction formats

Format I:

Opcode	r1	r2
4 bits	6 bits	6 bits

Format II:

Opcode	r1	r2	addr
4 bits	6 bits	6 bits	16 bits

Format III:

Opcode	r	data
10 bits	6 bits	16 bits

Example Machine 5:

Instructions in Format I:

Opcode	r1	r2
4 bits	6 bits	6 bits

Opcode	Mnemonic	Operation
0000	mov r1,r2	r1 $\leftarrow$ (r2);
0001	add r1,r2	r1 $\leftarrow$ (r1) + (r2);
0010	sub r1,r2	r1 $\leftarrow$ (r1) - (r2);
0011	and r1,r2	r1 $\leftarrow$ (r1) $\wedge$ (r2);
0100	or r1,r2	r1 $\leftarrow$ (r1) $\vee$ (r2);
0101	xor r1,r2	r1 $\leftarrow$ (r1) $\oplus$ (r2);

Example Machine 5:

Instructions in Format II:

Opcode	r1	r2	addr
4 bits	6 bits	6 bits	16 bits

Opcode	Mnemonic		Operation
0110	mov	r1,addr(r2)	r1 ← ($M_{addr+(r2)}$);
0111	mov	addr(r2),r1	$M_{addr+(r2)}$ ← (r1);
1000	mov	r1,r2,addr	r1 ← (r2); r2 ←(M_{addr});
1001	mov	addr,r1,r2	M_{addr} ← (r1); r1 ← (r2);
1010	je	r1,r2,addr	if (r1)=(r2) then PC ← addr;
1011	jne	r1,r2,addr	if (r1)≠(r2) then PC ← addr;
1100	jl	r1,r2,addr	if (r1)<(r2) then PC ← addr;
1101	jh	r1,r2,addr	if (r1)>(r2) then PC ← addr;
1110	call	r1,addr(r2)	r1←(PC); PC ← addr+(r2);

Example Machine 5:

Instructions in Format III:

Opcode	r	data
10 bits	6 bits	16 bits

Opcode	Mnemonic		Operation
1111000000	mov	r1,#data	r1 ← data;
1111000001	add	r1,#data	r1 ← (r1) + data;
1111000010	sub	r1,#data	r1 ← (r1) - data;
1111000011	and	r1,#data	r1 ← (r1) ∧ data;
1111000100	or	r1,#data	r1 ← (r1) ∨ data;
1111000101	xor	r1,#data	r1 ← (r1) ⊕ data;

Example Machine 5:

1) Determine the length of machine registers.

Answer:

L_{MAR} = 16 bits L_{MBR} = 16 bits L_{PC} = 16 bits

L_{IR} = 32 bits $L_{R0..R63}$ = 16 bits

2) Write a program to summate elements of an array of 100 elements. Translate your assembly program into machine code.

3) Write a program to accumulate the content of R0, R1, …, R62 into R62. Translate your assembly program into machine code. (**Homework #1**)

Example Machine 5:

2) Write a program to summate elements of an array of 100 elements. Translate your assembly program into machine code.

```
        org   100h

        xor   R0,R0         ; accumulator
        mov   R1,R0         ; index register
        mov   R2,#200       ; last element's index for loop control

loop:   mov   R3,A(R1)      ; loading the i-th element of A
        add   R0,R3         ; adding it to accumulator
        add   R1,#2         ; advancing the index register for next element
        jne   R1,R2,loop    ; if index is not 200 then continue

        mov   sum,R0,R0     ; store the result into variable sum

        call  R0,0(R63)     ; return to OS. Assume R63 has return address

sum:    dw    0             ; variable to keep summation
A:      dw    100 dup(?)    ; definition of a 100-element array
        end                 ; end of program
```

Example Machine 5:

2) Write a program to summate elements of an array of 100 elements. Translate your assembly program into machine code.

Address	Code (H)	Assembly Line			
0100			org	100h	
0100	5000		xor	R0,R0	0101 000000 000000
0102	0040		mov	R1,R0	0000 000001 000000
0104	F00200C8		mov	R2,#200	1111000000 000010 [00C8h]
0108	60C10120	loop:	mov	R3,A(R1)	0110 000011 000001 [0120h]
010C	1003		add	R0,R3	0001 000000 000011
010E	F0410002		add	R1,#2	1111000001 000001 [0002h]
0112	B0420108		jne	R1,R2,loop	1011 000001 000010 [0108h]
0116	9000011E		mov	sum,R0,R0	1001 000000 000000 [011Eh]
011A	E03F0000		call	R0,0(R63)	1110 000000 111111 [0000h]
011E	0000	sum:	dw	0	
0120	??????????	A:	dw	100 dup(?)	
			end		

Addressing modes

1. **Direct**
2. **Indirect**
3. **Immediate**
4. **Indexed**
5. **Implied/Inherent**: Nothing in IF but CPU knows where is the operand!

Example 1: loop instruction in 80806/88 processor.

loop address ; cx $\leftarrow$ (cx)-1;
; if (cx)≠0 then PC $\leftarrow$address;

Example 2: Multiply register instruction in IBM360 family.

MR 2,4 ; R_2:R_3 $\leftarrow$ (R_3) x (R_4)

One-address Machine

One operand is kept in a special purpose register named ACC (Accumulator) and addressed inherently.

Example: In a one-address machine:
- main memory size = 2^{16} addressable units (each 4 bits)
- word size is 16 bits (unaligned, big endian)
- Implied, direct, indirect, and immediate addressing modes

Instructions are coded in one format.

Opcode	addr/data
4 bits	16 bits

One-Address Machine

Opcode	addr/data
4 bits	16 bits

Opcode	Mnemonic		Operation
0000	load	#data	ACC $\leftarrow$ data;
0001	add	#data	ACC $\leftarrow$(ACC)+data;
0010	load	addr	ACC $\leftarrow$(M_{addr});
0011	store	addr	$M_{addr}$$\leftarrow$(ACC);
0100	add	addr	ACC $\leftarrow$ (ACC) + (M_{addr});
0101	sub	addr	ACC $\leftarrow$ (ACC) - (M_{addr});
0110	and	addr	ACC $\leftarrow$ (ACC) $\wedge$ (M_{addr});
0111	or	addr	ACC $\leftarrow$ (ACC) $\vee$ (M_{addr});
1000	xor	addr	ACC $\leftarrow$ (ACC) $\oplus$ (M_{addr});
1001	load	(addr)	ACC $\leftarrow$($M_{(M\_addr)}$);
1010	store	(addr)	$M_{(M\_addr)}$$\leftarrow$ (ACC);
1011	j	addr	PC $\leftarrow$ addr;
1100	jz	addr	if (ACC)=0 then PC$\leftarrow$addr;
1101	jnz	addr	if (ACC)$\neq$0 then PC$\leftarrow$addr;
1110	jn	addr	if (ACC)<0 then PC$\leftarrow$addr;
1111	jp	addr	if (ACC)>0 then PC$\leftarrow$addr;

One-address Machine

Write a program to generate first 20 elements of Fibonacci series and store them in an array.

```
        org     0
loop:   load    (pointer_1)
        store   sum
        load    (pointer_2)
        add     sum
        store   (pointer_3)
        load    pointer_1
        add     #4
        store   pointer_1
        add     #4
        store   pointer_2
        add     #4
        store   pointer_3
        load    counter
        add     #-1
        store   counter
        jnz     loop
        j       OS_return_addr

pointer_1:  dw  array
pointer_2:  dw  array+4
pointer_3:  dw  array+8
counter:    dw  18
sum:        dw  ?
array:      dw  0,1,18 dup(?)
            end
```

Homework #2:
Translate this program to machine code.

Zero-address Machine (Stack Machine)

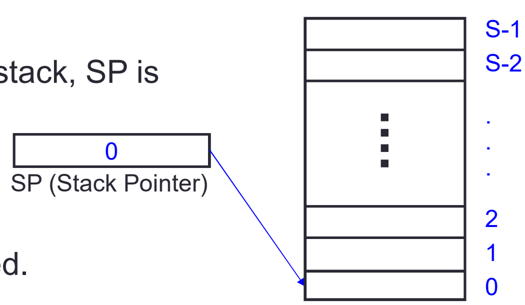
All operands are kept in a hardware stack inside CPU.

Hardware Stack: an internal memory of S words that follows LIFO (last in first out) strategy for writing (push) and reading (pop) data.

SP (Stack Pointer) register points to top of stack where it is ready to get new data.

When data is **push**ed into stack, SP is incremented.

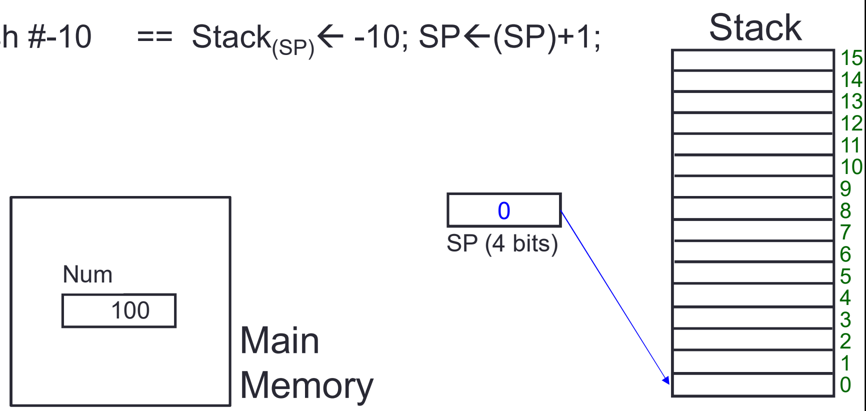
For **pop**, the SP is decremented, the word addressed by SP is returned.



Zero-address Machine (Stack Machine)

Example: Suppose a hardware stack of 16 words. SP is reset to 0 indicating that stack is empty. Perform the following push and pop operations. Num is a variable in memory initialize by 100.

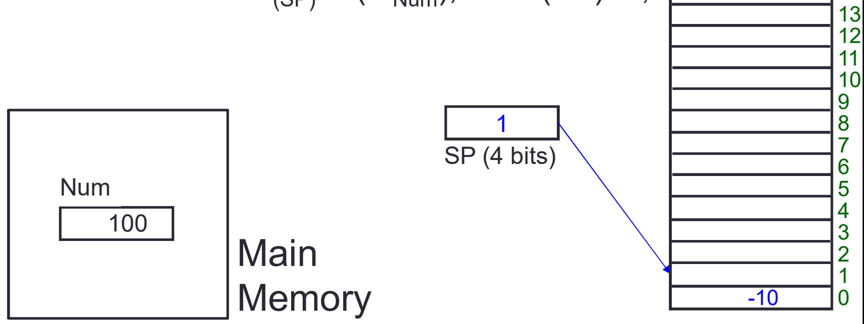
push #-10 == $Stack_{(SP)} \leftarrow -10; SP \leftarrow (SP)+1;$



Zero-address Machine (Stack Machine)

Example: Suppose a hardware stack of 16 words. SP is reset to 0 indicating that stack is empty. Perform the following push and pop operations. Num is a variable in memory initialize by 100.

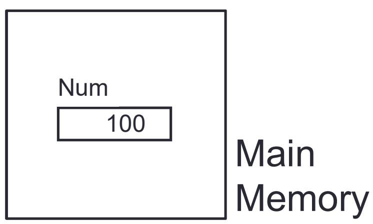
push #-10 == $Stack_{(SP)} \leftarrow -10; SP \leftarrow (SP)+1;$
push Num == $Stack_{(SP)} \leftarrow (M_{Num}); SP \leftarrow (SP)+1;$

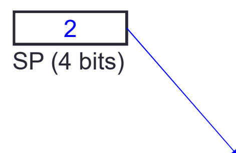


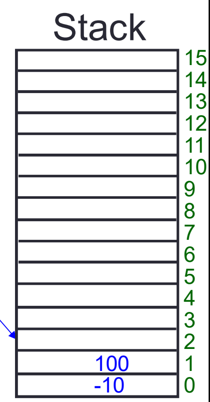
Zero-address Machine (Stack Machine)

Example: Suppose a hardware stack of 16 words. SP is reset to 0 indicating that stack is empty. Perform the following push and pop operations. Num is a variable in memory initialize by 100.

push #-10 == $Stack_{(SP)} \leftarrow -10$; $SP \leftarrow (SP)+1$;
push Num == $Stack_{(SP)} \leftarrow (M_{Num})$; $SP \leftarrow (SP)+1$;
push #77 == $Stack_{(SP)} \leftarrow 77$; $SP \leftarrow (SP)+1$;



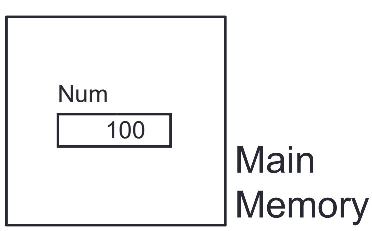


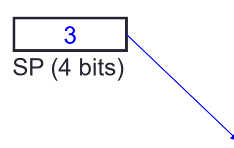


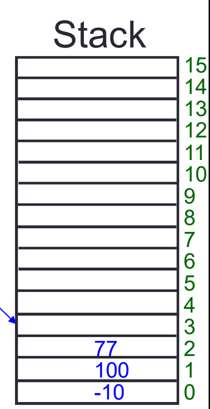
Zero-address Machine (Stack Machine)

Example: Suppose a hardware stack of 16 words. SP is reset to 0 indicating that stack is empty. Perform the following push and pop operations. Num is a variable in memory initialize by 100.

push #-10 == $Stack_{(SP)} \leftarrow -10$; $SP \leftarrow (SP)+1$;
push Num == $Stack_{(SP)} \leftarrow (M_{Num})$; $SP \leftarrow (SP)+1$;
push #77 == $Stack_{(SP)} \leftarrow 77$; $SP \leftarrow (SP)+1$;
pop Num == $SP \leftarrow (SP)-1$; $M_{num} \leftarrow (Stack_{(SP)})$;



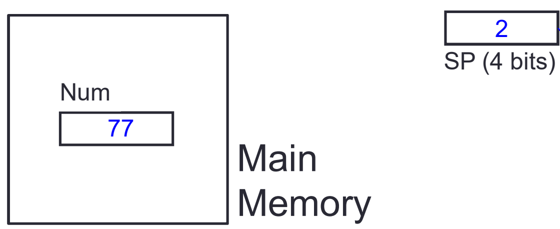


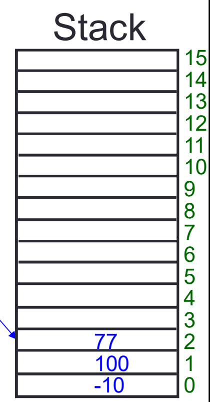


Zero-address Machine (Stack Machine)

Example: Suppose a hardware stack of 16 words. SP is reset to 0 indicating that stack is empty. Perform the following push and pop operations. Num is a variable in memory initialize by 100.

push #-10 == $Stack_{(SP)} \leftarrow -10$; $SP \leftarrow (SP)+1$;
push Num == $Stack_{(SP)} \leftarrow (M_{Num})$; $SP \leftarrow (SP)+1$;
push #77 == $Stack_{(SP)} \leftarrow 77$; $SP \leftarrow (SP)+1$;
pop Num == $SP \leftarrow (SP)-1$; $M_{num} \leftarrow (Stack_{(SP)})$;



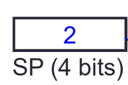


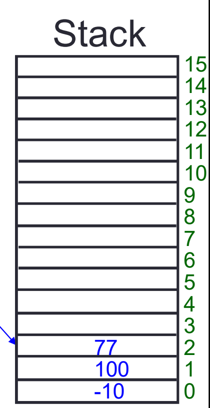
Zero-address Machine (Stack Machine)

To execute a function, say **func(a,b)**, the operands **a** and **b** are taken from top of stack and the result **func(a,b)** is pushed into stack.

func == pop temp1;
pop temp2;
temp3 $\leftarrow$ func ((temp1) , (temp2));
push temp3

Example: add



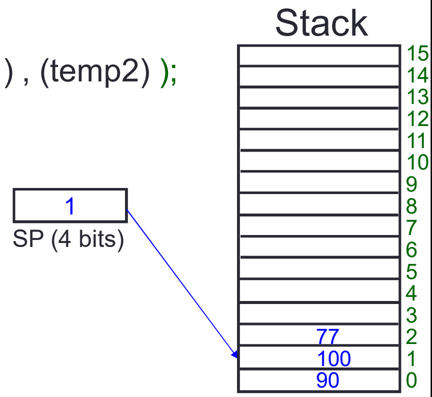


Zero-address Machine (Stack Machine)

To execute a function, say **func(a,b)**, the operands **a** and **b** are taken from top of stack and the result **func(a,b)** is pushed into stack.

```
func  ==  pop    temp1;
          pop    temp2;
          temp3 ← func ( (temp1) , (temp2) );
          push   temp3
```

Example: add



Polish Notation and Stack Machines

The normal notation we use in maths is best known as infix notation where the operator comes between operands.

Example: x + y or z * t

But alternative notation can be used to write mathematical expressions, e.g. prefix notation as:

+ x y or * z t

Postfix notation (so called Polish notation) as:

x y + or z t *

Polish Notation and Stack Machines

Example: Write the following expression in Polish notation:

$$\frac{x + y - 10 - t / d + 10*s - z}{20*t - z / t + 100*w}$$

Solution:

xy+10-td/-10s*+z-20t*zt/-100w*+/

Positive point:
- No need to worry about priority of operations. No parentheses.
- Suitable for calculation on a stack machine.

Negative point:
- Not good for human perception.

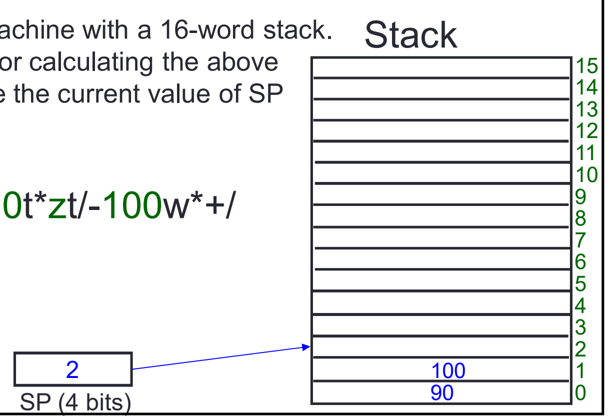
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/



Polish Notation and Stack Machines

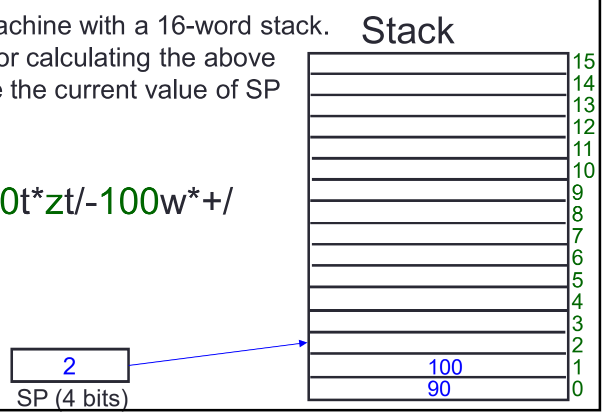
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push x



Polish Notation and Stack Machines

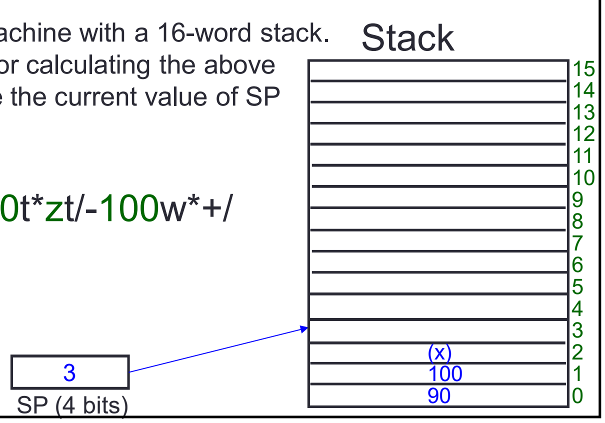
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push x



Polish Notation and Stack Machines

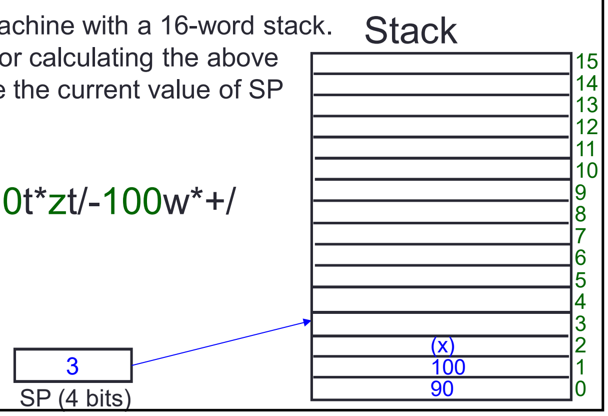
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push y



Polish Notation and Stack Machines

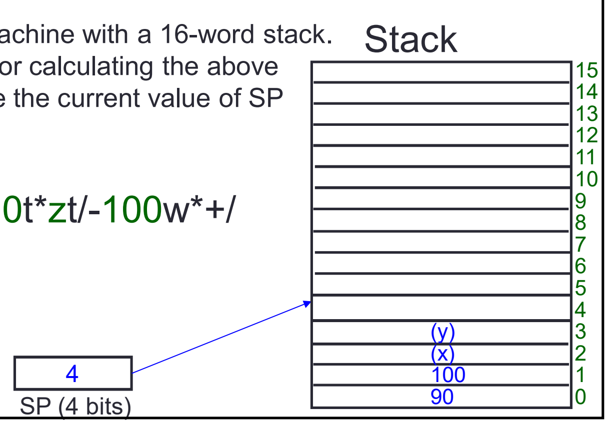
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push y



Polish Notation and Stack Machines

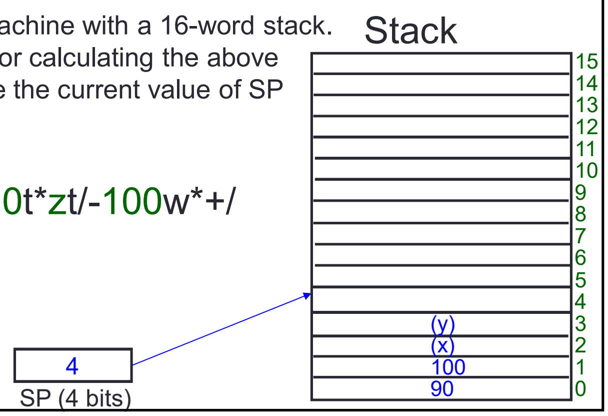
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Add



Polish Notation and Stack Machines

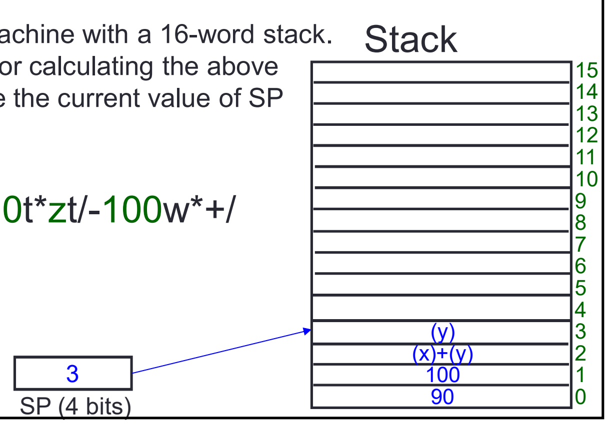
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Add



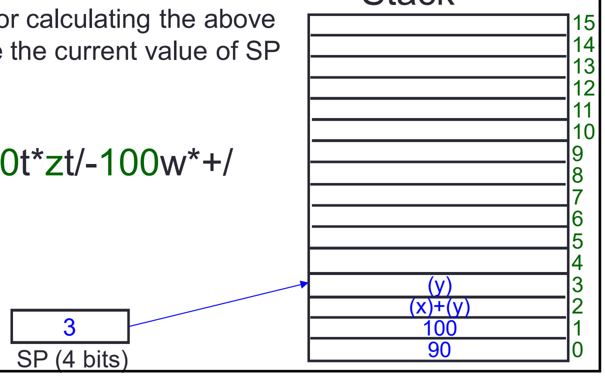
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

Stack



xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 10

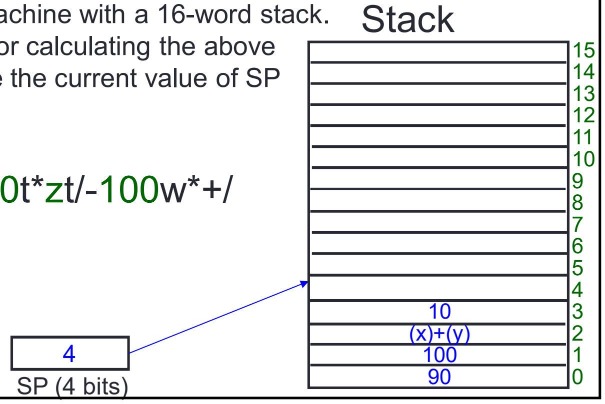
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

Stack



xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 10

Polish Notation and Stack Machines

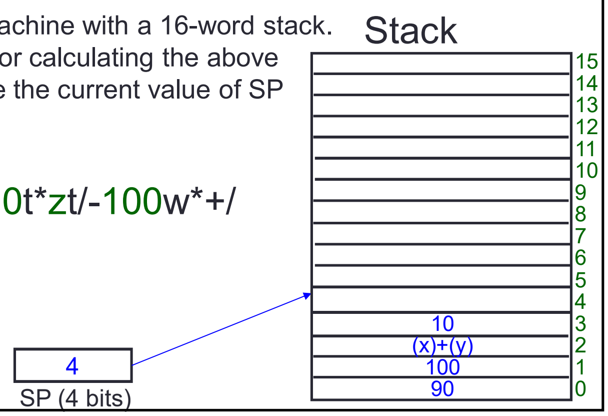
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub



Polish Notation and Stack Machines

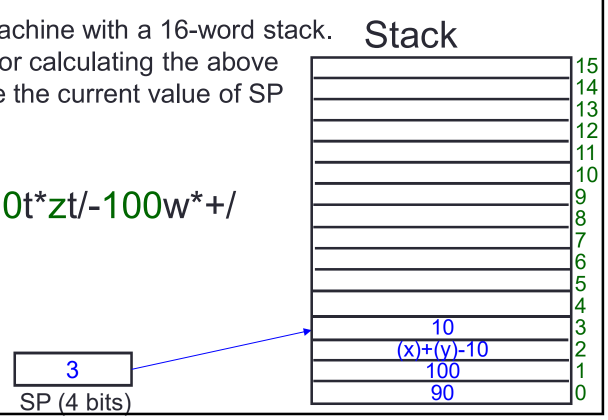
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub



Polish Notation and Stack Machines

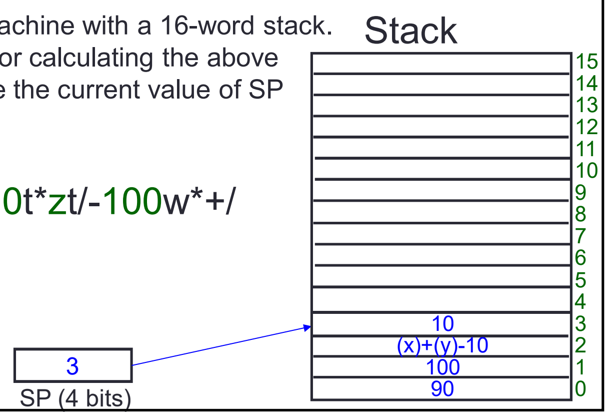
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push t



Polish Notation and Stack Machines

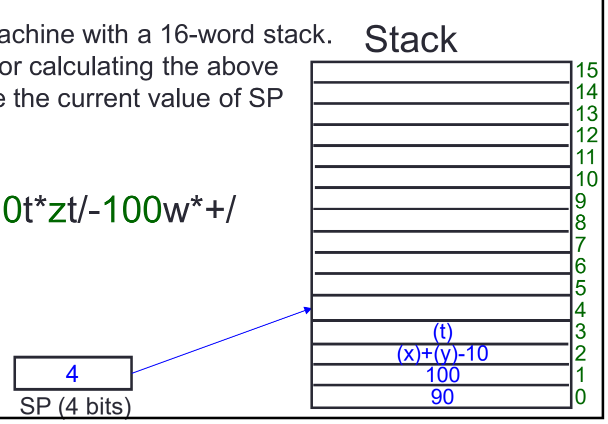
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push t



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#3: Registers, Addressing modes, Linker, Loader 75

Polish Notation and Stack Machines

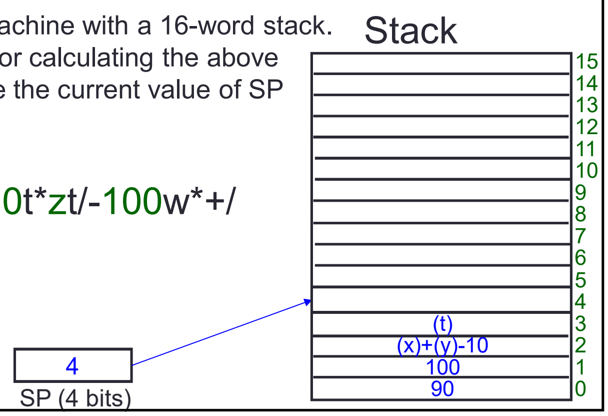
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push d



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#3: Registers, Addressing modes, Linker, Loader 76

Polish Notation and Stack Machines

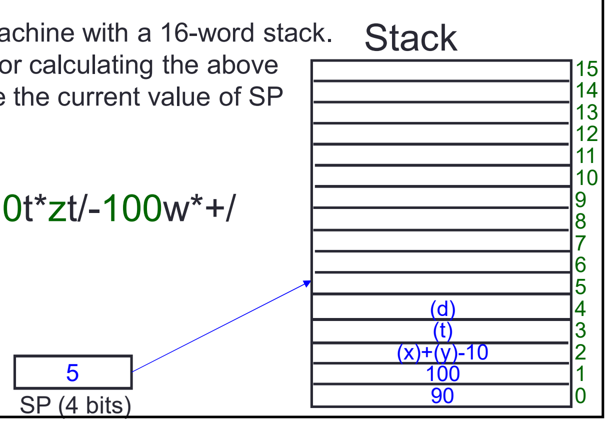
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push d



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

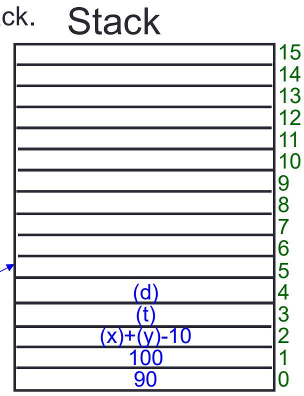
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Div

5

SP (4 bits)



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

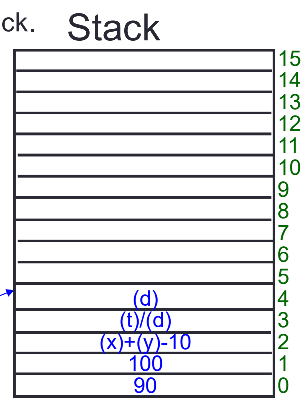
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Div

4

SP (4 bits)



Polish Notation and Stack Machines

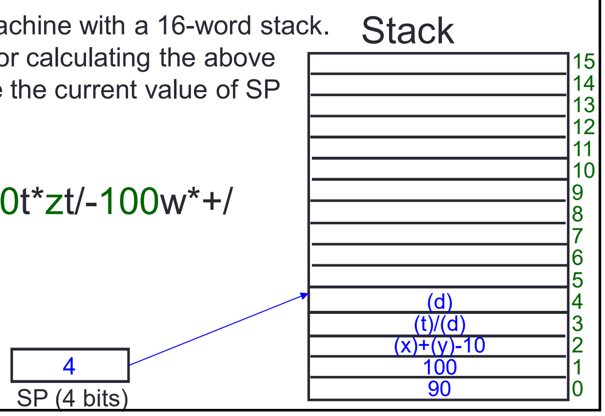
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub



Polish Notation and Stack Machines

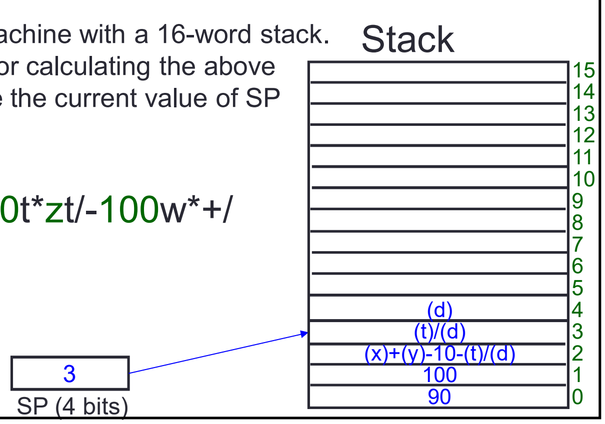
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub



Polish Notation and Stack Machines

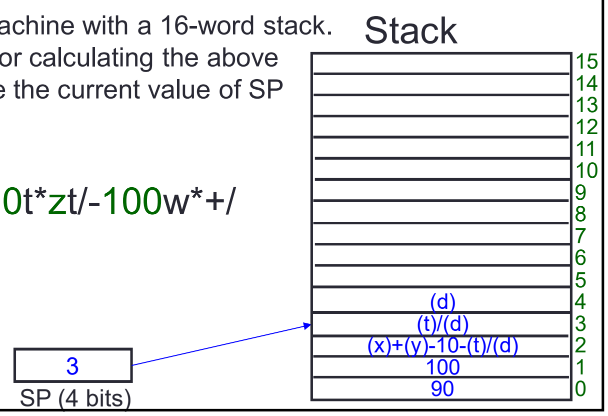
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 10



Polish Notation and Stack Machines

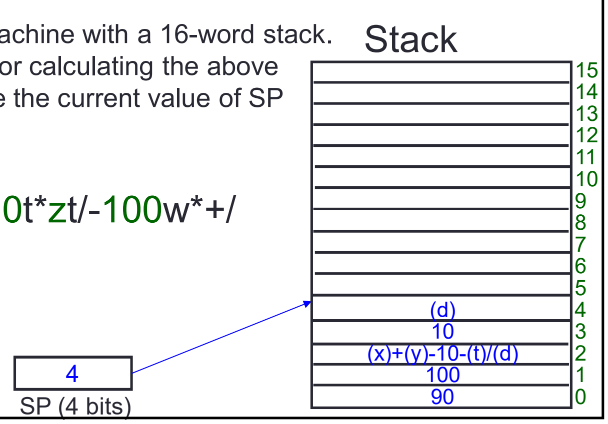
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 10



Polish Notation and Stack Machines

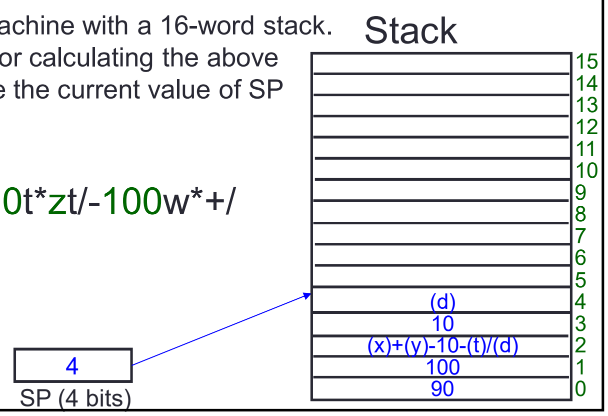
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push s



Polish Notation and Stack Machines

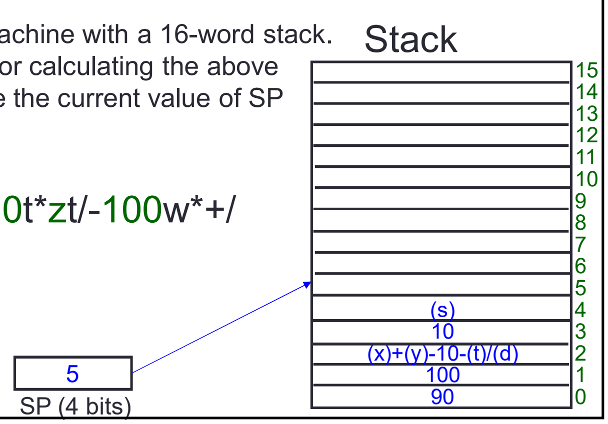
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push s



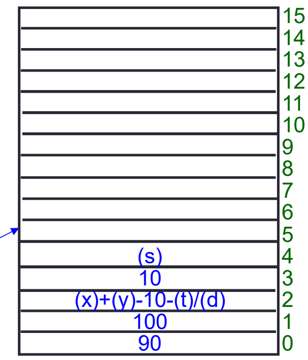
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

Stack



xy+10-td/-10s*+z-20t*zt/-100w*+/

Mult

5

SP (4 bits)

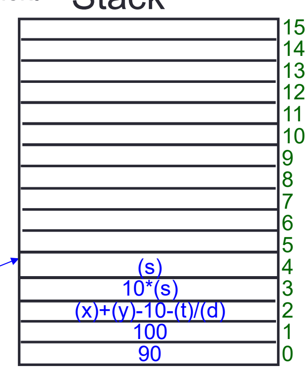
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

Stack



xy+10-td/-10s*+z-20t*zt/-100w*+/

Mult

4

SP (4 bits)

Polish Notation and Stack Machines

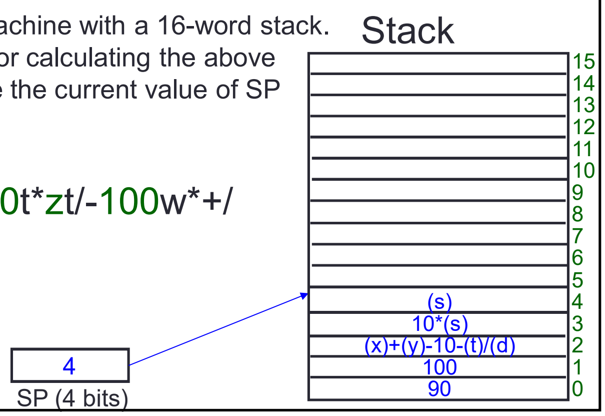
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Add



Polish Notation and Stack Machines

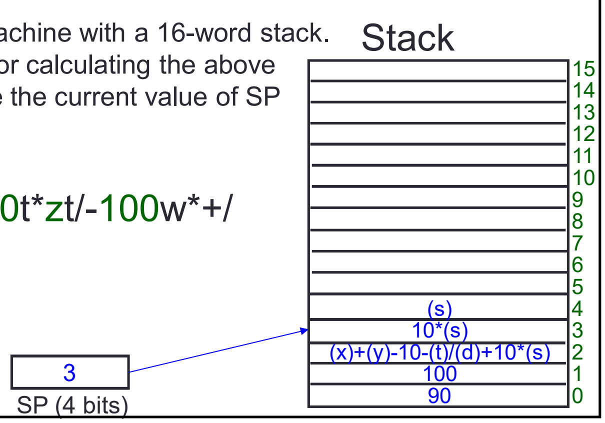
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Add



Polish Notation and Stack Machines

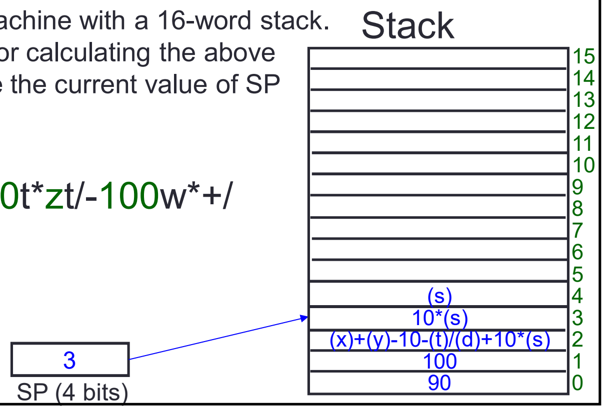
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push z



Polish Notation and Stack Machines

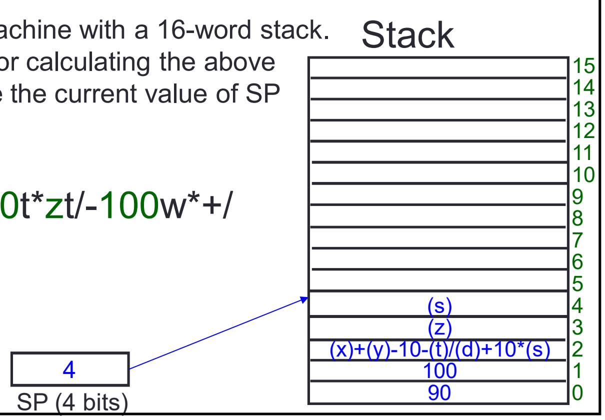
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push z



Polish Notation and Stack Machines

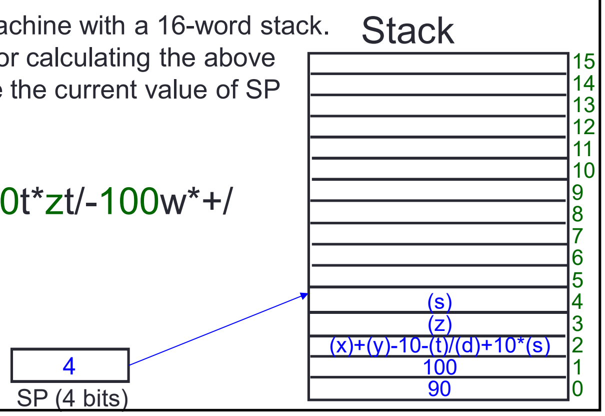
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub



Polish Notation and Stack Machines

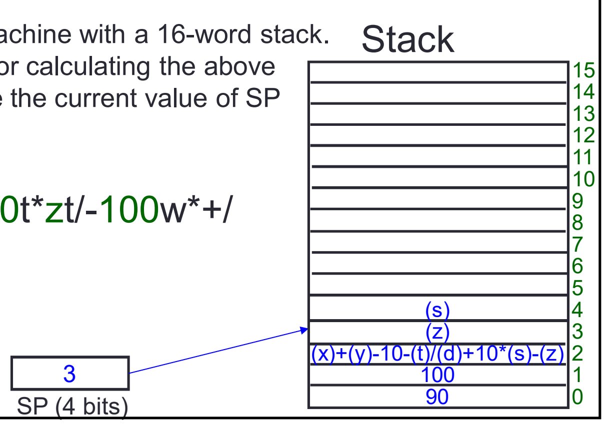
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

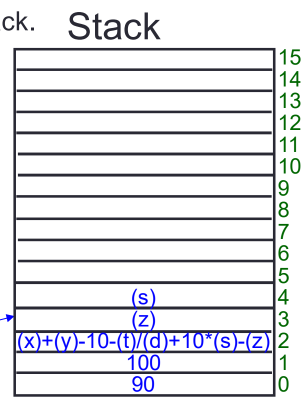
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 20

3

SP (4 bits)



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

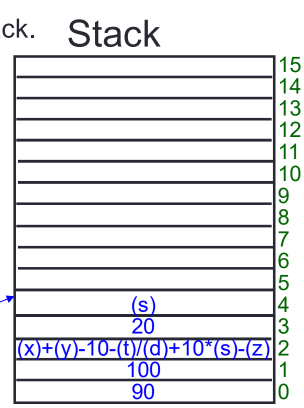
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 20

4

SP (4 bits)



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

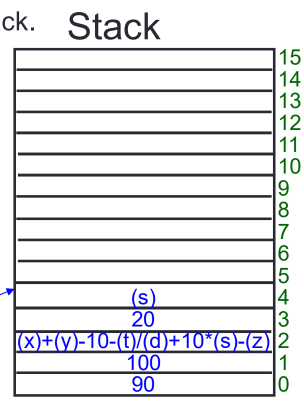
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push t

4

SP (4 bits)



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

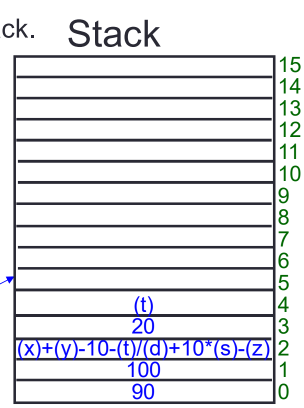
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push t

5

SP (4 bits)

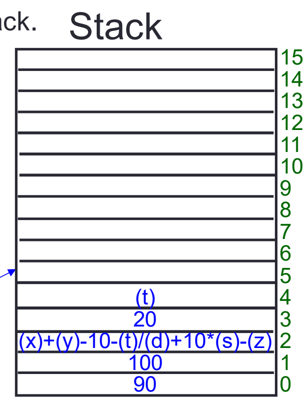


Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.



xy+10-td/-10s*+z-20t*zt/-100w*+/

Mult

5

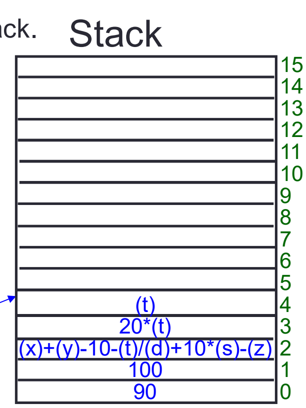
SP (4 bits)

Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.



xy+10-td/-10s*+z-20t*zt/-100w*+/

Mult

4

SP (4 bits)

Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

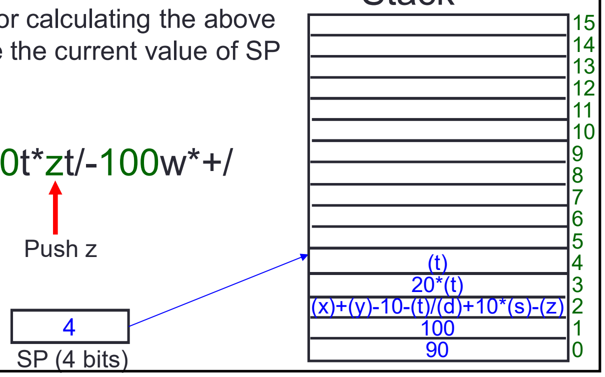
Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

Stack

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push z



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

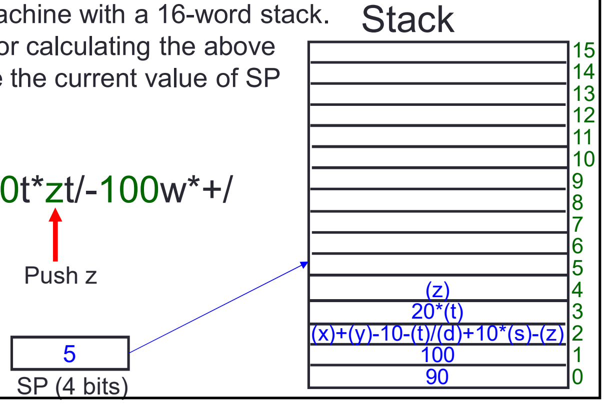
Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

Stack

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push z



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

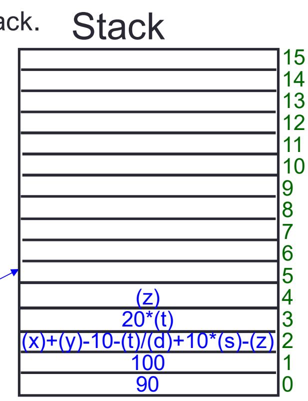
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push t

5

SP (4 bits)



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

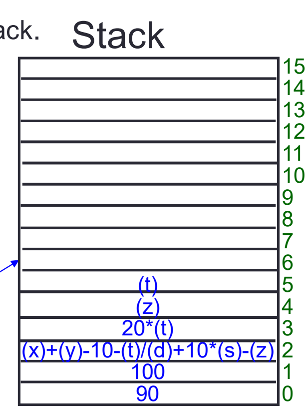
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push t

6

SP (4 bits)



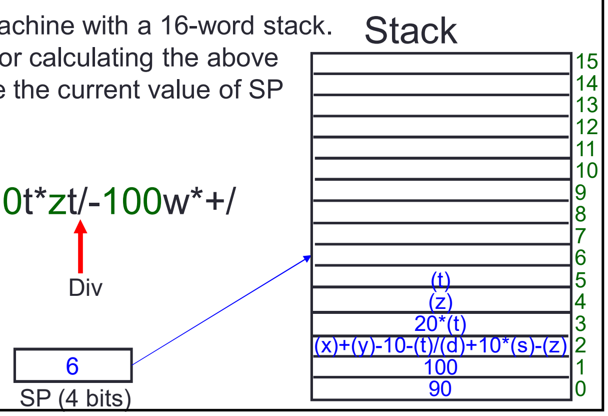
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/



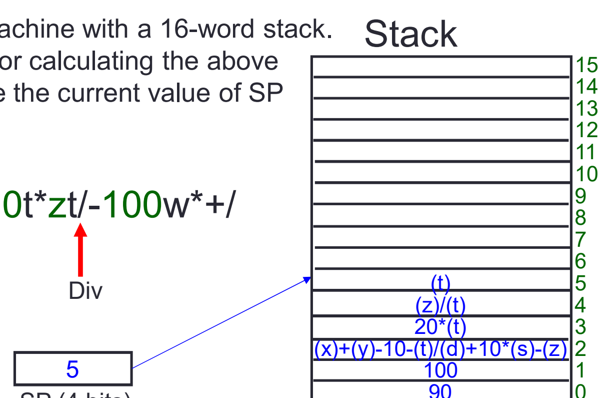
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

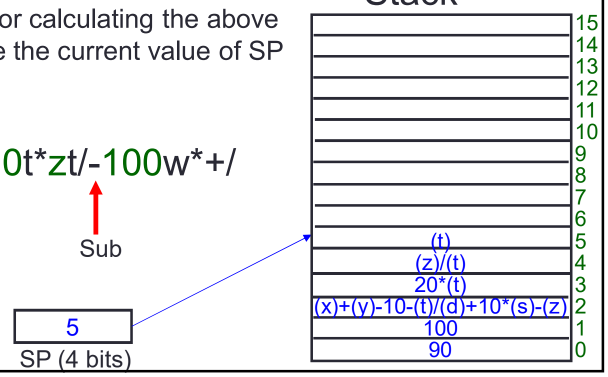
Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub

Stack



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

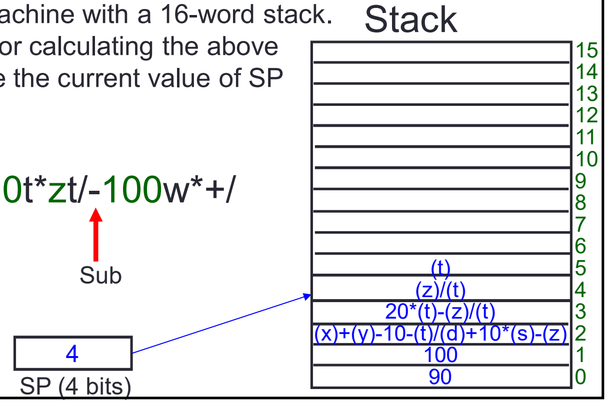
Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Sub

Stack



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

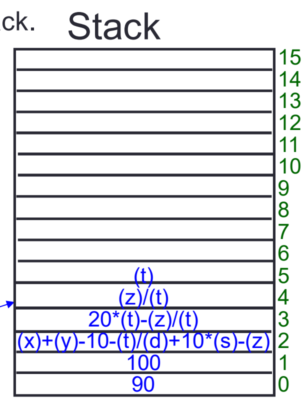
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 100

4

SP (4 bits)



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#3: Registers, Addressing modes, Linker, Loader 108

Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

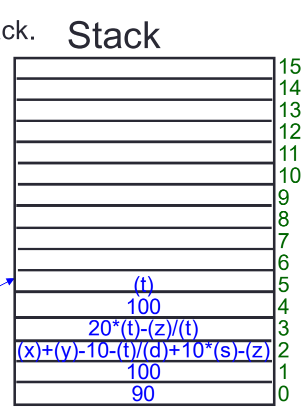
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push 100

5

SP (4 bits)



Polish Notation and Stack Machines

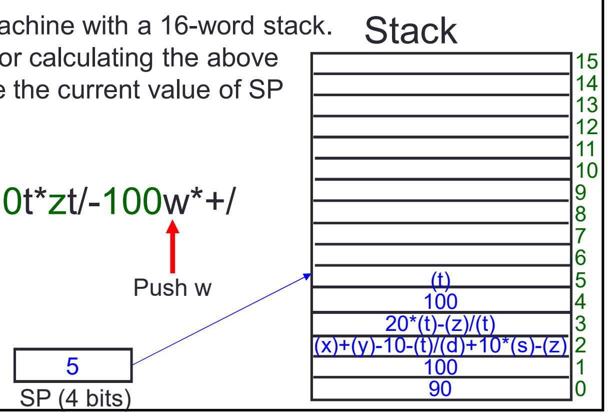
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push w



Polish Notation and Stack Machines

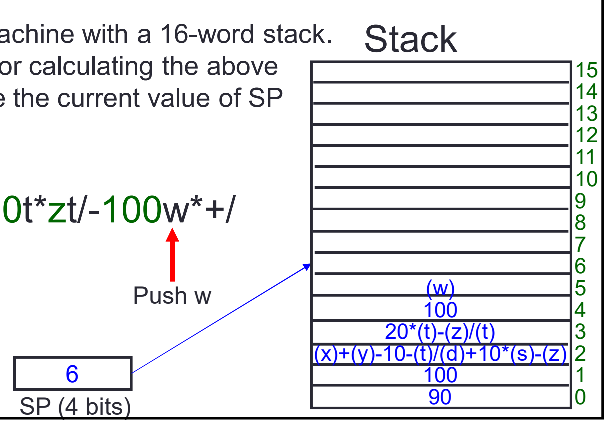
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Push w



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

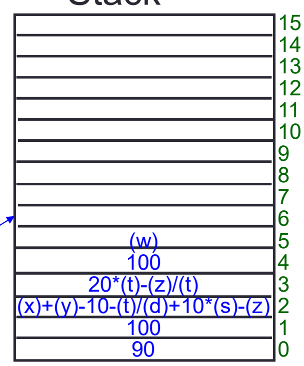
Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Mult

Stack



6

SP (4 bits)

Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

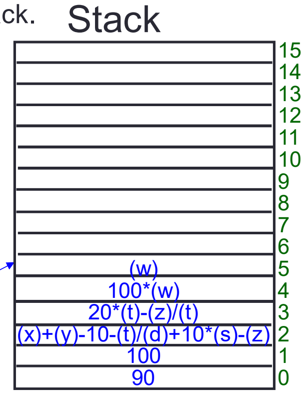
Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Mult

Stack



5

SP (4 bits)

Polish Notation and Stack Machines

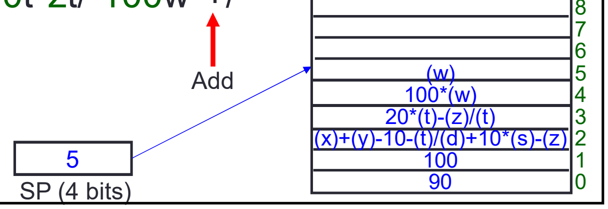
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Stack



Polish Notation and Stack Machines

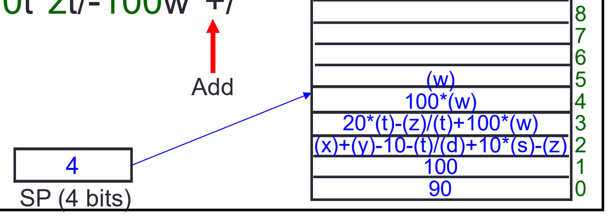
Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Stack



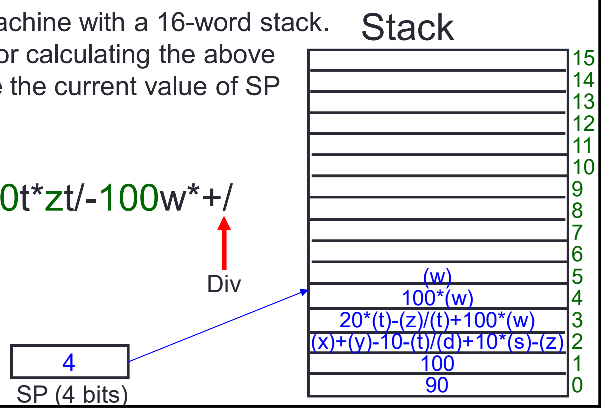
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/



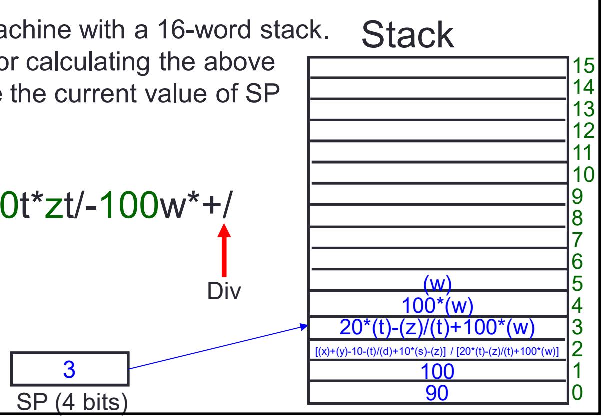
Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

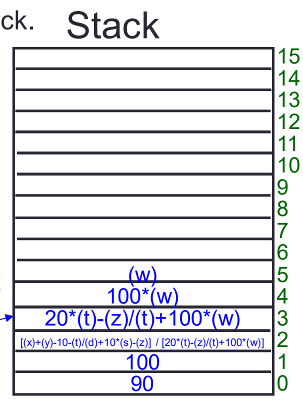
Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Pop Result

3

SP (4 bits)



Polish Notation and Stack Machines

Lets show the calculation of xy+10-td/-10s*+z-20t*zt/-100w*+/ in a stack machine.

Procedure: Scan the expression from left to right. If you see an operand just push it into stack. If you see an operator just perform it.

Lets assume we have a machine with a 16-word stack. Let us see what happens for calculating the above Polish expression. Assume the current value of SP is 2.

xy+10-td/-10s*+z-20t*zt/-100w*+/

Result

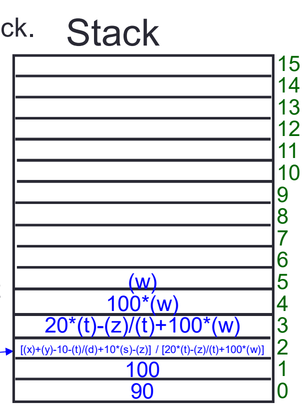
[(x)+(y)-10-(t)/(d)+10*(s)-(z)] / [20*(t)-(z)/(t)+100*(w)]

Main Memory

Pop Result

2

SP (4 bits)



(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#3: Registers, Addressing modes, Linker, Loader 119

Zero-address Machine (Stack Machine)

In a stack machine, all instructions (except push and pop) have no operand fields in the instruction format. So, almost all instructions have no operand fields in the instruction format.

Example Machine 6: In a stack machine:
- main memory size = 2^{32} addressable units (each 8 bits)
- word size is 32 bits
- Implied, direct, immediate addressing modes

Instructions are coded in two formats.

Format I

Opcode	Addr/data
8 bits	32 bits

Format II

Opcode
8 bits

(c) Hamid Sarbazi-Azad Computer Structure & Language -- Lecture#3: Registers, Addressing modes, Linker, Loader 120

Example Machine 6

Opcode	Addr/data
8 bits	32 bits

Format I

Opcode	Mnemonic	Operation
00000000	push #data	$Stack_{(SP)} \leftarrow$ data; $SP \leftarrow (SP)+1$;
00000001	push addr	$Stack_{(SP)} \leftarrow (M_{addr})$; $SP \leftarrow (SP)+1$;
00000010	pop addr	$SP \leftarrow (SP)-1$; $M_{addr} \leftarrow (Stack_{(SP)})$;

Opcode
8 bits

Format II

Opcode	Mnemonic	Operation
00000011	add	$Stack_{(SP)-2} \leftarrow (Stack_{(SP)-2})+(Stack_{(SP)-1})$; $SP \leftarrow (SP)-1$;
00000100	sub	$Stack_{(SP)-2} \leftarrow (Stack_{(SP)-2}) - (Stack_{(SP)-1})$; $SP \leftarrow (SP)-1$;
00000101	swap	$Stack_{(SP)-1} \leftarrow (Stack_{(SP)-2})$ and $Stack_{(SP)-2} \leftarrow (Stack_{(SP)-1})$;
00000110	sp++	$SP \leftarrow (SP)+1$;
00000111	sp--	$SP \leftarrow (SP)-1$;
00001000	j	$PC \leftarrow (Stack_{(SP)-1})$; $SP \leftarrow (SP)-1$;
00001001	jz	if $(Stack_{(SP)-2})=0$ then $PC \leftarrow (Stack_{(SP)-1})$; $SP \leftarrow (SP)-2$;
00001010	jnz	if $(Stack_{(SP)-2}) \neq 0$ then $PC \leftarrow (Stack_{(SP)-1})$; $SP \leftarrow (SP)-2$;
00001000	jn	if $(Stack_{(SP)-2})<0$ then $PC \leftarrow (Stack_{(SP)-1})$; $SP \leftarrow (SP)-2$;

Example Machine 6

Write a program to generate the sum of first 20 elements of Fibonacci series and store it in variable **sum**.

```
        org     0
loop:   push    x
        push    y
        add
        pop     x
        sp++
        push    sum
        add
        pop     sum
        push    x
        push    y
        add
        pop     y
        sp++
        push    sum
        add
        pop     sum
        push    counter
        push    #1
        sub
        pop     counter
        sp++
        push    #loop
        jnz
        push    #OS_return_addr
        j

sum:            dw      1
counter:        dw      10
x:              dw      0
y:              dw      1
                end
```

Homework: Generate the machine code of the above assembly program. Can you write a better program?

Infix to Postfix Conversion

The algorithm includes two steps:

1. Fully parenthesize the input expression.
2. Starting from inner pairs of parentheses, replace the right parenthesis with operator and delete the left parenthesis.

Example: Convert below expression into Polish notation:

$$\frac{x + y - 10 - t / d + 10*s - z}{20*t - z / t + 100*w}$$

Infix to Postfix Conversion

Example: Convert below expression into Polish notation:

$$\frac{x + y - 10 - t / d + 10*s - z}{20*t - z / t + 100*w}$$

Step 1:

(x+y-10-t/d+10*s-z) / (20*t-z/t+100*w)

((x+y)-10-(t/d)+(10*s)-z) / ((20*t)-(z/t)+(100*w))

(((x+y)-10)-(t/d)+((10*s)-z)) / (((20*t)-(z/t))+(100*w))

((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w))

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))

(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))
((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))
(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))
((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))
(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))
((xy+ 10-   td/- + 10s*z-) /   20t* zt/- 100w*+)

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))
((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))
(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))
((xy+ 10- td/- + 10s*z-) / 20t* zt/- 100w*+)

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))
((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))
(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))
((xy+ 10- td/- + 10s*z-) / 20t* zt/- 100w*+)
(xy+ 10- td/- 10s*z- + / 20t* zt/- 100w*+)

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))
((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))
(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))
((xy+ 10- td/- + 10s*z-) / 20t* zt/- 100w*+)
(xy+ 10- td/- 10s*z- + / 20t* zt/- 100w*+)

Infix to Postfix Conversion

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))

Step 2:

(((((x+y)-10)-(t/d))+((10*s)-z)) / (((20*t)-(z/t))+(100*w)))
((((xy+ -10)- td/)+(10s* -z)) / ((20t* - zt/)+ 100w*))
(((xy+ 10- - td/)+ 10s*z-) / (20t* zt/- + 100w*))
((xy+ 10- td/- + 10s*z-) / 20t* zt/- 100w*+)
(xy+ 10- td/- 10s*z- + / 20t* zt/- 100w*+)
xy+ 10- td/- 10s*z- + 20t* zt/- 100w*+ /

xy+10-td/-10s*z-+20t*zt/-100w*+/

END OF SLIDES