Y86

- The Y86 is
- A simple subset of the IA32(Intel) architecture.
  - The IA32 has
    - Too many instructions with an overly complicated encodeing
    - Too many addressing modes
    - Efficiency over simplicity
  - So we use a simplified version
- It has
  - 8 general-purpose 32-bit registers.
  - 1 program counter(PC)
  - 3 single-bit condition codes: ZF, SF, OF
  - 12 types of instruction

Y86 Instructions

- register/memory transfers:
  - rmmovl rA, D[rB] M_4[D + R[rB]] ← R[rA]
    - Example: rmmovl %edx, $0x20(%esi)
  - mrmovl D(rB), rA   R[rA] ← M_4[D + R[rB]]
    - Example: mrmovl $0x10(%edx), %esi
- Other data transfer instructions
  - rrmovl rA, rB   R[rB] ← R[rA]
  - irmovl V, rB   R[rB] ← V
- Arithmetic instructions(operate on registers only)
  - addl rA, rB R[rB] ← R[rB]+R[rA]
  - subl rA, rB R[rB] ← R[rB]-R[rA]
  - andl rA, rB R[rB] ← R[rB]&R[rA]
  - xorl rA, rB R[rB] ← R[rB]^R[rA]
  - mull rA, rB R[rB] ← R[rB]*R[rA]
  - divl rA, rB R[rB] ← R[rB]/R[rA]
  - divl rA, rB R[rB] ← R[rB]%R[rA]

Y86 Instructions cont’

- Unconditional
    - jmp Dest   PC <- Dest
- Conditional jumps
    - jle Dest   PC ← Dest if last result <=0
    - jl   Dest   PC ← Dest if last result ≤ 0
    - je   Dest   PC ← Dest if last result < 0
    - jne Dest   PC ← Dest if last result ≠ 0
    - jge Dest   PC ← Dest if last result ≥ 0
    - jg   Dest   PC ← Dest if last result > 0
- Conditional
    - cmovle rA, rB   R[rB] ← R[rA] if last result ≤ 0
    - cmovl   rA, rB   R[rB] ← R[rA] if last result < 0
    - cmove   rA, rB   R[rB] ← R[rA] if last result = 0
    - cmovne rA, rB   R[rB] ← R[rA] if last result ≠ 0
    - cmovge rA, rB   R[rB] ← R[rA] if last result ≥ 0
    - cmovg   rA, rB   R[rB] ← R[rA] if last result > 0

Example

- C Example

```
Int start[] = { 4,7,8,9,12,11};
int sum_function () {
        int sum = 0;
        int count = 6;
        int *str=start;

        while ( count) {
        sum += *str;
        str++;
        count--;
        }
        return sum;
}
```

```
IA32 Assembly:
sum_function:
movl $start, %edx          # int *str = start

xorl %eax, %eax            # int sum = 0

.L2:
addl (%rdx), %eax          #sum += *str

addq $4, %rdx              #str++, count––

cmpq $start+24, %rdx       #if count != 0

jne .L2                    #loop

rep
ret
```

```
Y86:
.pos 0x100
sum_function:
irmovl $start, %edx
xorl %eax %eax
L2:
mrmovl (%edx), %ebx
addl %ebx %eax
irmovl $4, %ebx
addl %ebx, %edx
irmovl $end, %ebx
subl %edx, %ebx
jne L2
ret
.pos 0x200
start:
. . .
end:
```

Y86 instruction set encoding

Byte	0		1		2	3	4	5
halt	0	0						
nop	1	0						
rrmovl **rA, rB**	2	0	rA	rB				
irmovl **V, rB**	3	0	F	rB	V			
rmmovl **rA, D(rB)**	4	0	rA	rB	D			
mrmovl **D(rB), rA**	5	0	rA	rB	D			
OPl **rA, rB**	6	fn	rA	rB				
jXX **Dest**	7	fn	Dest					
cmovXX **rA, rB**	2	fn	rA	rB				
call **Dest**	8	0	Dest					
ret	9	0						
pushl **rA**	A	0	rA	F				
popl **rA**	B	0	rA	F				

Number	Register Name
0	%eax
1	%ecx
2	%edx
3	%ebx
4	%esp
5	%ebp
6	%esi
7	%edi
F	No register

Program register identifiers

fn	jump
0	jmp
1	jle
2	jl
3	je
4	jne
5	jge
6	jg

7 jump functions

fn	operation
0	addl
1	subl
2	andl
3	xorl

Operations supported

Operations

addl	6	0
subl	6	1
andl	6	2
xorl	6	3

Branches

jmp	7	0	jne	7	4
jle	7	1	jge	7	5
jl	7	2	jg	7	6
je	7	3			

Moves

rrmovl	2	0	cmovne	2	4
cmovle	2	1	cmovge	2	5
cmovl	2	2	cmovg	2	6
cmove	2	3			