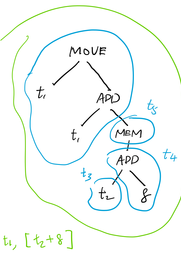
x86 and Instruction Selection

Tiling cover AST w/ tiles
add temps between tiles
emit code bottom-up.



MOVE(t_1, t_1 + MEM(t_2 + 8))

```
mov t3, t2
mov t4, t3
add t4, 8
mov t5, [t4]
add t1, t5
```

add t_1, [t_2 + 8]

x86 ISA: 2-address CISC (complex) ISA.

opcode dest&src, src

mov add sub mul ... cmp and or xor ... shl shr sar ...
jmp jz je jg jnp jle ... push pop call ret ...

Registers 8 32-bit gp registers

	16 bits	
	8 bits	8 bits
eax	ah	al
ebx	bh	bl
ecx	ch	cl
edx	dh	dl
esi		
edi		
esp		
ebp		

ax bx cx dx

Operands

	Intel	AT&T	TIR
constant	42	$42	CONST(42)
register	eax	%eax	TEMP(eax)
mem address	[eax]	(%eax)	MEM(TEMP(eax))
	[eax+32]	32(%eax)	
	[eax+ebx*4+7]	7(%eax, %ebx, 4)	

[base + index * scale + displacement]
gp reg {1,2,4,8} integer

byte 8 bits
word 16 bits
dword 32 bits

size directives / size inference

```
add eax, [edx]      32-bit inferred
inc dword [eax]
```

at most one mem operand per instruction.

```
add [ebx], ecx
add [eax], [ecx]    X not allowed
```

Branching

j cc cc = z|nz|l|le|g|ge|nge|o|no|p|pe|up|...

```
cmp eax ebx     computes eax - ebx and sets condition flags
jz L1           jump if zero
je
jl
jnge
```

set cc dest test cc and set 8 bits dest to 0 or 1

```
setz al
```

Tiles.

Define T[[e]]t (t: abstract register)
T[[s]]

gcc -O2 -S -masm=intel

T[[MOVE(t, 0)]] = mov t, 0

T[[MOVE(t, 0)]] = xor t, t — better

T[[ADD(t_1, t_2)]] t_3 =
```
mov t3, t1
add t3, t2
```

T[[ADD(t_1, t_2)]] t_3 = lea t_3, [t_1 + t_2] — better

T[[ADD(e_1, e_2)]] t = T[[e_1]] t_1 ; T[[e_2]] t_2 ; lea t_3, [t_1 + t_2]

T[[ADD(k, ADD(MUL(e_2, w)))]] t
(w: 1, 2, 4, 8)

= T[[e_1]] t_1 ; T[[e_2]] t_2 ; lea t, [t_1 + t_2 × w + k]

T[[EQ(e_1, e_2)]] t =
```
T[[e1]] t1 ; T[[e2]] t2 ;
cmp t1, t2 ;
je lt
mov t, 0
jmp l
lt: mov t, 1
l:
```

T[[EQ(e_1, e_2)]] t =
```
T[[e1]] t1 ; T[[e2]] t2 ;
cmp t1, t2 ;
setz al ;
movzx t, al
```

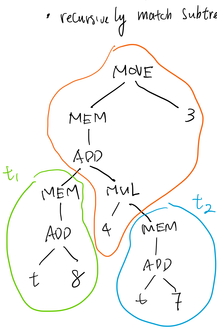
T[[CJUMP(e, l)]] =
```
T[[e]] t ;
test t, t     bitwise AND
jnz l
```

T[[CJUMP(EQ(e_1, e_2), l]] =
```
T[[e1]] t1 ; T[[e2]] t2 ;
cmp t1, t2 ;
je l
```

Algorithm

Greedy algorithm -

- Match root of IR AST with highest-priority matching tile
- recursively match subtrees




```
mov t1, [t + 8]
mov t2, [t + 7]
mov [t1 + t2 x 4], 3
```

Optimal tiling

Cost of tiling a tree using a tile = cost of tile + cost of tiling the subtrees

$$Cost(tr) = \min_{tile} \left(Cost(tile) + \sum_{tr'} Cost(tr') \right)$$

1. Use optimal tiling if memoized already
2. For each tile matching the root:
   - recursively tile subtrees
   - compute cost of tiling
3. Remember best tiling and it cost.

$O(n \cdot t)$