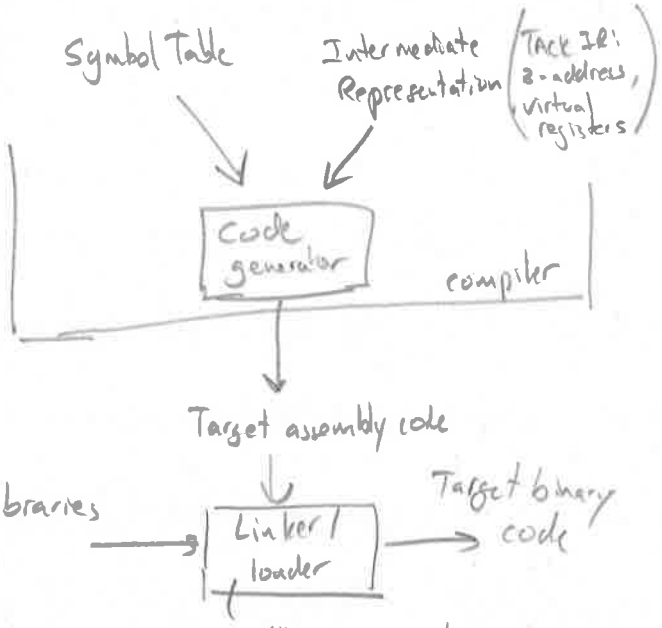
Code generation pg.1

Lecture Topics

I. Introduction to code generation
II. Target language x64
III. Code generation without register allocation

I. Introduction to Code Generation



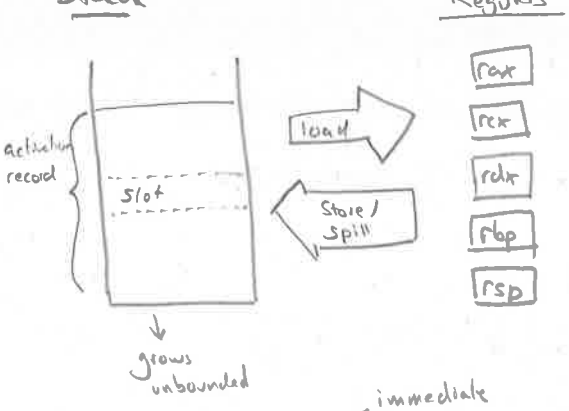
gcc -m64 -masm=intel main.s

	IR	Assembly	Binary
Instruction	Name	Name	Number
Variable	Name	Register or static offset	Register or static offset
Label	Name	Name	Number

Different kinds of assembly:

	CISC	RISC	Stack
Description	Complex Instruction Set computer	Reduced Instruction Set computer	Stack (virtual) machine
Example	x64	ARM	Java Bytecode
Instructions	Many, powerful	Few, small	Few, small
Addresses per Inst.	2	3	0
Registers	Few, non-uniform	Many, uniform	None
Memory Instructions	Most	Few	Stack: most others: few

Stack vs. registers



Addressing modes: immediate, register, memory

add r,i | add r,r | add r,m | add m,i | add m,r

but not
add m,m

Code generator has two tasks:
- instruction selection
- register allocation

Trade-offs:

Simple, fast compiler — slow target code

Complex, slow compiler — fast target code

II Target language x64

See x64-intro.pdf
- reading Figure 1
- running Figure 1
- experimenting with gcc
  - hello.c
  - loop.c
  - record.c
  - call.c

III Code generation without register allocation

For each function:

Preparation:

for each variable or temporary x:

x.offset = new offset

for each string constant s:

string[s] = new label

Code generation:

generate prologue

for each IR instruction:

load operands

"template" of x64 instructions

store result

generate epilogue

Examples:

IR:

```
x = y + z;
```

x64

```
mov rax, [rbp - y.offset]
add rax, [rbp - z.offset]
mov [rbp - x.offset], rax
```

IR:

```
if x goto l;
```

x64

```
cmp [rbp - x.offset], 0
jne l
```

Arrays

IR:

```
x = y[z]
```

x64

```
mov  rax, [rbp - y.offset]
mov  rdx, [rax + 8]
mov  rax, [rbp - z.offset]
sal  rax, 3
add  rax, rdx
mov  rax, [rax]
mov  [rbp - x.offset], rax
```