2013-10-22

Code Generation

Which computer are we aiming to compile for?

**Description of hardware:**

"Simple hardware"

Imagine a machine implemented in hardware.

*Code* = [array of bytes]. --> Filled up by compiler and executed by the runtime.

*Data* = [array of bytes] --> Where variables live. Would be allocated by the compiler. Physically present only during execution.

*Stack* = [array of ?] --> Used during execution. More than way of doing this. One way is making a byte array.

*Heap = [for dynamic allocation]. Not used in this course*

"Registers"

IP Instruction Pointer Current Position in Code

SP Stack Pointer Current Position in Stack

BP (FP) base / frame ptr

=================================================================================

***Examples***

var x,y:integer;

c:char;

These reside in the data array but only physically exist only during execution.

x would be at 0, y would be at 4 and c at 8.

Total data size needed = 9;

This is assuming that integer takes 4 bytes and char 1byte.

--------------------------------------------------------------------------------------------------------------------

Initialized variables

int x=3,y=5;

char c='a';

one option is to create code to initialize the variables another is to let the compiler create the data with the variables initialize.

Not to worry about this since pascal doesn't have this.

-------------------------------------------------------------------------------------------------------------

x + y

push x

push y

add

====================================================================================

**Instructions**

Some will be added to this list later in the course.

1 Byte opcode + Optional Operands

opcode is to determine what instruction we have to do.

**1. Stack Operations**

push addr

pushi value

pop addr

var x,y:integer;

x is defined @0

y is defined @4

x:= y + 5;

push 4

push 5

add

pop 0

===================================================================

Assume that variables and addresses are all 4bytes.

Representations would look like this:

Push addr #define cp.push 1

Push

pushi addr #define cp.pop 2

pop addr #define cp.pushi 3

void run(){

ip = 0;

while(1){

switch(code[ip++]){

case op\_push:

int a = \*(int \*)(code+ip); ip+=4;

int val = \*(int \*)(data + a);

stack[sp++].i = val;

break;

case op\_pushi:

int val = \*(int \*)(code+ip); ip+=4;

stack[sp++].i = val;

break;

case op\_pop:

int a = \*(int \*)(code+ip); ip+=4;

int val = stack[sp--].i = val;

\*(int \*)(data + a) = val;

break;

case op\_add:

stack[sp-2].i += stack[sp-1].i;

sp--;

break;

also sub,mul,div,and, or,xor,shl,shr

not and neg (unary)

fadd,fsub,fmul,fneg, fdiv

cvr, cvi //convert

eql, neg, ess, gtr, leq,geq,feql,fneg

(3. Control)

case stop:

exit();

break;

case jump:

int a = \*(int \*)(code+ip); ip+=4;

ip = a;

break;

case jumpTrue:

int a = \*(int \*)(code+ip); ip+=4;

if (stack[sp] == true){ ip = a;}

break;

case jumpFalse:

int a = \*(int \*)(code+ip); ip+=4;

if (stack[sp] == false){ ip = a;}

break;

also call and return

(4. I/O)

case op\_writeint:

val = stack[--sp];

cout << val; break;

}

}

}

=================================================================================

Build the byte array by hand.

see if we can execute the above code

==================================================================================

I/O

writeint

(output int from stack).

-----------------------------------------------------------------------------------------------------------------------

Code would be in a file on disk. code.c ====> code.class

Machine program would reside in a separate file.\

Stack should be variable push2, push4 push1