Assignment 3

WebAssembly to x86

COP701: Software Systems Laboratory

Nilaksh Agarwal
Indian Institute of Technology, Delhi
2015PH10813
ph1150813@iitd.ac.in

Navreet Kaur
Indian Institute of Technology, Delhi
2015TT10917
tt1150917@iitd.ac.in

I. Memory Allocation

Each function is assumed to occupy 4kB, with 1kB allocated to the stack and 3kB to the array. This allocation is indexed by the function number in webassembly. We assume a 4kB array for the global variables (indexed before function 0)
The stack contains all elements which are accessed in x86 through registers. All unary operators apply on the topmost element of the stack and all binary operators on the top two. The stack elements are indexed at the base location of the function and occupy 4 bytes each. The array is a 3kB sized data structure index starting from function memory + 1kB offset due to the stack. The indices of this array are mapped to the LocalIndex obtained from local.get and local.set of Wasabi.

II. WebAssembly to x86 instruction mapping

Some of the instructions are straightforward and can be directly mapped to their corresponding instruction in x86 format. These are given in tabular form below. For other instructions, a brief method for mapping is given for each of them.

A. Numeric Instructions (Simple Instructions)
These are included under *unary*, *binary*, *const_, select,* and *drop* hooks.

WebAssembly Instruction	**x86 Instruction**
i32.const	EQU
i32.clz	LZCNT
i32.add	ADD

i32.sub	SUB
i32.mul	IMUL
i32.eq	CMP
i32.trunc_f32_u	VCVTSS2USI
i32.trunc_f64_s	CVTTSD2SI
i64.trunc_f32_s	CVTTSS2SI
i64.trunc_f32_u	CVTTSS2SI
f32.abs	FABS
f32.neg	NEG
f32.sqrt	FSQRT
f32.demote_f64	CVTSD2SS
f32.convert_i32_s	VCVTUSI2SS
f64.min	MINSD
f64.max	MAXSD

B. Control Instructions (Simple Instructions)
These are included in *nop, unreachable, br, br_if, begin, end, return_,* and *call* hooks.

WebAssembly Instruction	**x86 Instruction**
nop	NOP
unreachable	UD2
block	BEGIN
loop	LOOP
br	JMP
return	RET
end	END
call	CALL

1. br: `<pc> 4 <branch target pc>`
2. br_if: if condition if true, then `<pc> 4 <branch target pc>`, else `NOP`
3. end:
    a) if type of *end* is *loop*, then `<pc> 27 LOOP <loop address>`

b) if type of end is *if*, then
`<pc> 27 JMP <difference between end address and current address>`
`<pc> 4 <end address>`

C. Parametric Instructions (Simple Instructions)

WebAssembly Instruction	x86 Instruction
drop	EQU
select	EQU

1. drop: `<pc> 27 <store address> EQU 0`, and decrease the size of stack by 1
2. select: `<pc> 27 <store address> EQU <value>`, and increase stack size by 1

D. Variable Instructions (Load and Store Instructions)

WebAssembly Instruction	x86 Instruction
local.get	LW
local.set	SW
global.get	LW
glocal.set	SW
local.tee	SW

1.

E. Memory Instructions
The below table shows the ideal mapping for the following instructions.
However, since we are not implementing a register stack, instead of the below mapping, we have used LW for *load* and SW for *store* everywhere.

WebAssembly Instruction	x86 Instruction
i32.load	FILD

i32.store	FIST
i64.load	FILD
i64.store	FIST
i32.load8_s	FILD
i32.load16_s	FILD
f32.load	FLD
f32.store	FST
f64.load	FLD
f64.store	FST

F. Structured Instructions (Branch taken/not-taken instructions)
This includes the operations included under the *if_* hook.
We first the address of where the branch points. The location is uniquely determined by *func* and *instr*.

If the condition is true, then the branch is not taken:
`<pc> 27 JMP <difference between the end address and the current address>`
`<pc>  5 <end address>`

If the condition is not true, then the branch is taken:
`<pc> 27 JMP <difference between the end address and the current address>`
`<pc> 4 <end address>`

III. **Usage Instructions**

```
sh run.sh <trace size> <filename.wasm>
```

The output of this code is a file with name <filename.gz>