Assembly Language Lecture Series: Fundamentals of RISC-V Assembly Language

Roger Luis Uy
College of Computer Studies
De La Salle University
Manila, Philippines

Fundamentals of RISC-V Assembly

- Registers
- Instructions
- Pseudo-Instructions
- Assembler Directives
- Environment calls
- Macros

Fundamentals of RISC-V Assembly

- RISC-V website: https://riscv.org/
- RISC-V specifications and manual (vol 1): https://riscv.org/technical/specifications/
- RARS (RISC-V Assembler and Runtime simulator)

https://github.com/TheThirdOne/rars

- Latest version (jar file):

https://github.com/TheThirdOne/rars/releases/tag/v1.5

- Wiki:

https://github.com/TheThirdOne/rars/wiki

RISC-V Assembly registers

RISC-V registers (RV32I)

- Integer registers are 32-bit in length
- Floating-point registers are 32-bit (for single-precision) or 64-bit (for double precision)
- 32 integer registers + program counter ***pc***
- ***pc*** holds the address of the current instruction
- register **x1** to hold the return address for a call
- register **x5** can serve as an alternate return address
- register **x2** as the stack pointer

RISC-V integer registers

Register name	ABI name	Description	Owner
x0	zero	Always zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	thread pointer	
x5	t0	Temporary/alternate return address	Caller
x6 – x7	t1 - t2	Temporary	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function argument/return value	Caller
x12-x17	a2-a7	Function argument	Caller
x18-x27	s2-s11	Saved register	Callee
x28-x31	t3-t6	Temporary	Caller

Caller-saved registers (aka call-clobbered or volatile registers) are used to hold temporary quantities that need not be preserved across call.
If you want to keep the register's value you need to save it before you call a function

Callee-saved registers (aka call-preserved or non-volatile registers) are used to hold long-lived values that should be preserved across calls.
You need to save it before you overwrite it, when the function is being called.

RISC-V floating-point registers

Register name	ABI name	Description	Owner
f0-f7	ft0-7	FP temporaries	caller
f8-f9	fs0-1	FP saved registers	Callee
f10-f11	fa0-1	FP arguments/return values	Caller
f12-f17	fa2-7	FP arguments	Caller
f18-f27	fs2-11	FP saved registers	Callee
f28-f31	ft8-11	FP temporaries	Caller

Caller-saved registers (aka call-clobbered or volatile registers) are used to hold temporary quantities that need not be preserved across call.
If you want to keep the register's value you need to save it before you call a function

Callee-saved registers (aka call-preserved or non-volatile registers) are used to hold long-lived values that should be preserved across calls.
You need to save it before you overwrite it, when the function is being called.

RISC-V Assembly instructions

RISC-V base: RV32I instructions

- 42 unique instructions
- Can emulate almost any other ISA extensions
- Extensions:
- RV32G: M (mult/div), F (single float), D (double float)

RISC-V base: RV32I

Load and store instructions				
LB	LBU	LH	LHU	LW
SB	SH	SW		

Integer computation instructions (register-register)				
ADD	SLT	SLTU	AND	OR
XOR	SLL	SRL	SRA	SUB
NOP				

Integer computation instructions (register-immediate)				
ADDI	SLTI	SLTIU	ANDI	ORI
XORI	SLLI	SRLI	SRAI	LUI
AUIPC				

RISC-V base: RV32I

Control transfer (unconditional branch) instructions				
JAL	JALR			

Control transfer (conditional branch) instruction				
BEQ	BNE	BLT	BLTU	BGE
BGEU				

RISC-V base: RV32I

System instructions				
ECALL	EBREAK	FENCE	HINT	

RISC-V Assembly Pseudo-instructions

RISC-V Pseudo-instructions

- Pseudo-instructions are not part of the instruction set but can be directly replace with real instructions
- RISC instruction set minimizes instruction. Thus, some instructions that are not available can be made using other instructions.
- Example: there is no instruction to assign a constant to a register except to use it with an addi instruction → addi x5, x0, 20
- But there is a pseudo-instruction **li** (load immediate) which is used to assign constant to register → li x5, 20
- other pseudo-instruction example: mv x5, x6 (internally: add x5, x0, x6)

Not all pseudo-instructions are supported by RARS

Some RISC-V pseudo-instructions

Pseudo-instructions	Description
la rd, symbol	Load address
LB/LH/LW/LD *rd*, *symbol*	Load integer from memory to register
SB/SH/SW/SD *rd*, *symbol*	store integer from register to memory
li *rd*, *immediate*	Load immediate
mv *rd*, *rs*	Register to register transfer
fmv.s *fd*, *fs*	Single precision register to register transfer
fmv.d *fd*, *fs*	Double precision register to register transfer
beqz/bnez/blez/bgez/bltz/bgtz *rs*, *offset*	Branch if zero comparison
bgt, ble, bgtu, bleu *rs*, *rt*, *offset*	Branch if >, <= (signed and unsigned)
J *offset*	Jump (unconditional)
JAL *offset*	Jump and link
call *offset*	Jump and link (call far-away subroutine)
ret	Return from subroutine

RISC-V Assembly directives

RISC-V directives (non-comprehensive)

Directive	Description
.byte	Store the listed value(s) as 8-bit byte
.half	Store the listed value(s) as 16-bit half word
.word	Store the listed value(s) as 32-bit word
.dword	Store the listed value(s) as 64-bit double word
.float	Store the listed value(s) as 32-bit single precision
.double	Store the listed value(s) as 64-bit double precision
.space	Reserve the next specified number of bytes in data segment
.ascii	Store the string without null terminator
.asciz	Store the string with null terminator
.data	Data segment
.text	Code segment
.globl	Declare the listed label(s) as global to enable referencing from other files
.macro / .end_macro	Macro definition

RISC-V Assembly Environment call

Environment call/System call

- System call number in register a7
- other parameters into register a0 to a6
- then issue `ecall`

Environment call/System call

- System call: Exit (i.e., return 0)

```
li a7, 10     ; system call 10 is exit
ecall
```

Environment call/System call

- System call: Print string (a7 = 4)

```
.data
str: .asciz “Hello World”
.text
la a0, str    ; a0=address of the null-terminated string
li a7, 4      ; system call
ecall
```

Environment call/System call

- System call: Print char (a7 = 11)

```
li a0, 0x41   ; character to be printed (in ASCII)
li a7, 11     ; system call
ecall
```

Environment call/System call

- System call: Print signed integer (a7 = 1)

```
li a0, -1       ; signed integer to be printed
li a7, 1        ; system call
ecall
```

Environment call/System call

- System call: Print unsigned integer (a7 = 36)

```
li a0, 25     ; unsigned integer to be printed
li a7, 36     ; system call
ecall
```

Environment call/System call

- System call: Print hex (a7 = 34)

```
li a0, 25     ; integer to be printed in hex
li a7, 34     ; system call
ecall
```

Environment call/System call

- System call: Print binary (a7 = 34)

```
li a0, 25     ; integer to be printed in binary
li a7, 35     ; system call
ecall
```

Environment call/System call

- System call: Print float (a7 = 2)

```
.data
var1: .float 4.0
.text
la t1, var1  ; t1 points to var1
flw fa0,(t1) ; single-precision float to be printed
li a7, 2     ; system call
ecall
```

Environment call/System call

- System call: Get string (a7 = 8)

```
.data
str: .space 11    ; reserve 10 bytes
.text
la a0, str    ; a0=address of the buffer
li a1, 11     ; max string length
li a7, 8      ; system call
ecall
```

Environment call/System call

- System call: Get char (a7 = 12)

```
li a7, 12     ; system call (a0 = character)
ecall
```

Environment call/System call

- System call: Get integer (a7 = 5)

```
li a7, 5        ; system call (a0 – integer)
ecall
```

Environment call/System call

- System call: Get float (a7 = 6)

```
li a7, 6        ; system call (fa0 – float)
ecall
```

Environment call/System call

- System call: open file (a7 = 1024)
- System call: read file (a7 = 63)
- System call: write file (a7 = 64)
- System call: close file (a7 = 57)
- System call: Confirm dialog box (a7 = 50)
- System call: Message dialog box (a7 = 55)
- System call: Message dialog box with integer (a7 = 56)
- System call: Message dialog box with float (a7 = 58)
- System call: Message dialog box with string (a7 = 59)

RISC-V Assembly macro

Macro

- macro can be created using .macro …. .end_macro
- all macros can be store in one file (example: macros.asm)
- to use the macro file, add .include <*macro_file_name*> in your program

example (without parameter):

```
.macro NEWLINE
li a0, 10
li a7, 11
ecall
.end_macro
```

example (with parameter):

```
.macro PRINT_DEC (%x)
li a7, 1
mv a0, %x
ecall
.end_macro
```

.eqv directive

- .eqv directive is used to substitute an arbitrary string for an identifier

example:
.eqv CTR t2
.eqv LIMIT 20
.eqv CLEAR_CTR li CTR, 0

to use in the code:
CLEAR_CTR
add CTR, CTR, 1
--preprocess and translated as
addi t2, x0, 0
add t2, t2, 1