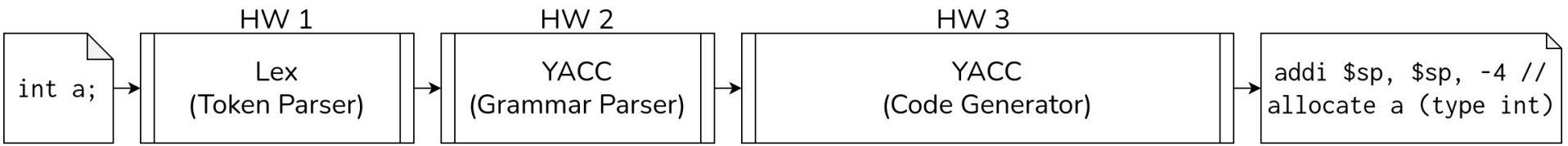
Intro to RISC-V

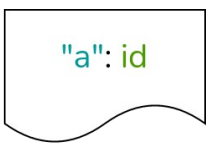
CS3404: Compiler Design

Outline

1. Overview of Homeworks
2. ISA and RISC-V
    1. The RISC-V ISA
    2. RISC-V Registers
    3. Common RISC-V Instructions
3. Calling Convention

1. Overview of Homeworks






```
var_decl {
  printf("addi $sp, $sp, -%d //
allocate %s (type %s)\n",
      sizeof(type.get()),
id.get(), type.get());
}
```

(Codes here are pseudo and only for the purpose of concept demonstration)

2-1. The RISC-V ISA

- Instruction Set Architecture (ISA)
  - Defines an **assembly language**, specifying available instructions, registers, ...
  - RISC (Reduced Instruction Set Computation) ISA vs CISC (Complex ISC) ISA
    - RISC: Only provides **simpler** instructions (e.g. add/sub registers, load/store, ...)
    - CISC: Provides both simple and **complex** instructions (e.g. add/sub memories, ...)
- RISC-V: A Free & Open RISC ISA
  - First proposed by UC Berkeley in 2010
  - Popular in Internet-of-Things devices
  - Allows IC designers to extend the ISA on their own

2-2. RISC-V Registers

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

2-2. Common RISC-V Instructions

- NOTE: Here we only cover the **“format”** of basic instructions. You should look up for concrete instructions from the standard (or a cheat sheet :)) on your own.
- Arithmetic Inst.
  - Format: `<operation> <dst_reg>, <src_reg_0>, <src_reg_1>`
  - Example: `add a0, a1, a2` performs `a0 = a1 + a2`
  - Note: Some of these has the ‘i’ variant that takes an immediate value as `<src_reg_1>`
    - Example: `addi sp, sp, -4` performs `sp = sp + (-4)`

2-2. Common RISC-V Inst. (cont.)

- Memory Load/Store Inst.
  - Format: `<l/s><size> <dst_reg> <imm_offset>(<base_reg>)`
    - Possible `<size>`: `b` (byte), `h` (halfword), `w` (word), `d` (doubleword)
    - `<imm_offset>`: signed offset relative to `<base_reg>`
  - Example
    - `lw a0, 12(sp)` loads a word at `memory[sp + 12]` into `a0`
    - `sb a1, -4(fp)` stores a byte from `a0` into `memory[fp + (-4)]`
  - Note: RISC-V is byte-addressed

2-2. Common RISC-V Inst. (cont.)

- Conditional Branch Inst.
  - Format: `b<cond> <pred_reg_0>, <pred_reg_1>, <imm_offset>`
    - Possible `<cond>`: `eq` (==), `ne` (!=), `ge` (signed >=), `geu` (unsigned >=), `lt` (signed <), `ltu` (unsigned <)
    - `<imm_offset>`: signed offset (relative to `pc`) with 2 bytes as unit
  - Example
    - `beq a0, a1, 4` sets `pc` to `pc + 8` if `a0 == a1`

2-2. Common RISC-V Inst. (cont.)

- Unconditional Branch Inst.
  - Jump and link: `jal <link_reg>, <imm_offset>`
    - `<imm_offset>`: signed offset (relative to `pc`) with 2 bytes as unit
    - Example: `jal x1, 40` stores `pc + 4` into `x1` and jumps to `pc + 80`
  - Jump and link register: `jalr <link_reg>, <base_reg>, <imm_offset>`
    - `<imm_offset>`: signed offset (relative to `<base_reg>`) with **single** byte as unit
    - Example: `jalr x1, fp, -128` stores `pc + 4` into `x1` and jumps to `fp - 128`

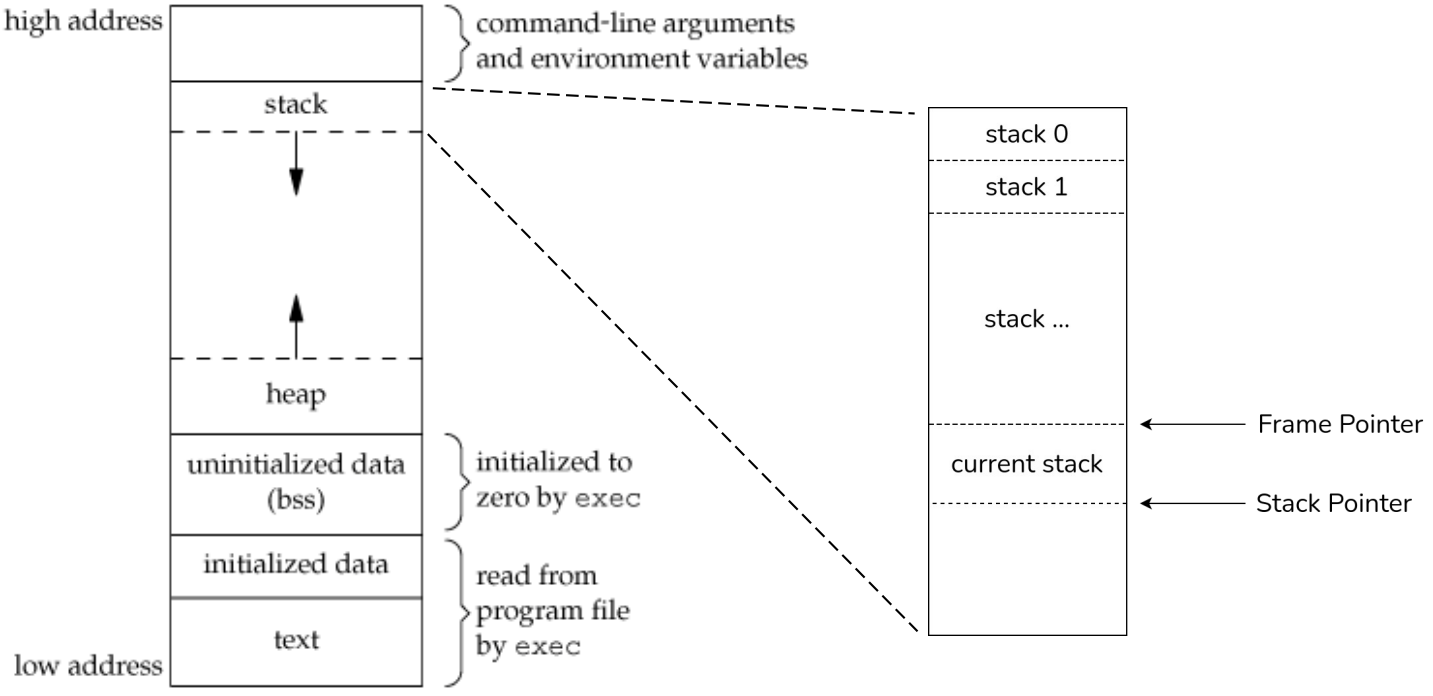
3. Calling Convention

- What are “calling convention” for?
  - A single program has only 1 set of registers and 1 memory space
  - ➔ Functions need to coordinate between themselves to manage these resources
  - ➔ The coordination is then known as “calling convention”, but it’s just a convention (not a standard)

3. Calling Convention (cont.)

- In HW 3, you'll compile a program containing multiple functions, thus the introduction of calling convention
- The RISC-V standard includes an official calling convention of register saving/restoration to facilitate
  - Creation and maintenance of frames in program stack
  - Passing function arguments
  - Returning values from function
  - ...

3. Calling Convention (cont.)



3. Calling Convention (cont.)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

- In a function call scenario, we have the caller and callee
  - E.g. in `void foo() { bar(); }`, `foo` is the caller, and `bar` is the callee
- The RISC-V official calling convention specifies saver of registers
  - To save a register, we push it onto the program stack
  - To restore, we pop it from the stack

3. Calling Convention (cont.)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Consider:

```
void foo() {
  /* do something ... */
  /* foo-pre-bar */
  bar();
  /* foo-post-bar */
  /* do some other things ... */
}
void bar() {
  /* bar-prologue */
  /* do something ... */
  /* bar-epilogue */
}
```

3. Calling Convention (cont.)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

When generating code for `foo()`, the compiler would insert codes that saves caller-saved registers at `/* foo-pre-bar */`:

```
sw ra, -4(sp)
sw t0, -8(sp)
sw t1, -12(sp)
sw t2, -16(sp)
sw a0, -20(sp)
sw a1, -24(sp)
sw a2, -28(sp)
sw a3, -32(sp)
sw a4, -36(sp)
sw a5, -40(sp)
sw a6, -44(sp)
sw a7, -48(sp)
sw t3, -52(sp)
sw t4, -56(sp)
sw t5, -60(sp)
sw t6, -64(sp)
addi sp, sp, -64
```

3. Calling Convention (cont.)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

The caller-saved registers get restored at `/* foo-post-bar */`:

```
addi sp, sp, 64
lw t6, -64(sp)
lw t5, -60(sp)
lw t4, -56(sp)
lw t3, -52(sp)
lw a7, -48(sp)
lw a6, -44(sp)
lw a5, -40(sp)
lw a4, -36(sp)
lw a3, -32(sp)
lw a2, -28(sp)
lw a1, -24(sp)
lw a0, -20(sp)
lw t2, -16(sp)
lw t1, -12(sp)
lw t0, -8(sp)
lw ra, -4(sp)
```

3. Calling Convention (cont.)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

When generating code for `bar()`, the compiler would insert codes that saves callee-saved registers at `/* bar-prologue */`:

```
sw s0, -4(sp)
addi sp, sp, -4
addi s0, sp, 0 // create new frame
sw sp, -4(s0)
sw s1, -8(s0)
sw s2, -12(s0)
sw s3, -16(s0)
sw s4, -20(s0)
sw s5, -24(s0)
sw s6, -28(s0)
sw s7, -32(s0)
sw s8, -36(s0)
sw s9, -40(s0)
sw s10, -44(s0)
sw s11, -48(s0)
addi sp, s0, -48
```

3. Calling Convention (cont.)

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

The callee-saved registers get restored at `/* bar-epilogue */`:

```
lw s11, -48(s0)
lw s10, -44(s0)
lw s9, -40(s0)
lw s8, -36(s0)
lw s7, -32(s0)
lw s6, -28(s0)
lw s5, -24(s0)
lw s4, -20(s0)
lw s3, -16(s0)
lw s2, -12(s0)
lw s1, -8(s0)
lw sp, -4(s0)
addi sp, sp, 4
lw s0, -4(sp) // restore frame
```

Thanks

Appendix

- The RISC-V Inst Cheat Sheet: https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCardv8-20151013.pdf