

RISC-V Assembly Programming

Muhammad Tahir

Lecture 6

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

① Assembly Programming

② Function Call Conventions

Data Processing Instructions

- Example data processing instructions and their descriptions

```
add    x3, x4, x5      // x3 = x4 + x5
addi   x3, x4, 9        // x3 = x4 + 9
xor     x1, x2, x3      // x1 = x2 ^ x3
slt     x2, x4, x6      // x2 = (x4 < x6) ? 1:0
```

```
// load upper immediate (lui) used to build 32-bit constants
lui     x3, imm         // x3 = imm << 12 (20-bit 'imm' value)

// add upper immediate to PC (auipc)
auipc   x4, imm         // x4 = PC + (imm << 12)
```

Data Processing Instructions Cont'd

- Generating 32-bit constants

```
// Generating 32-bit constant  
y = 0x12345678;
```

```
// Assume x6 = y  
lui    x6, 0x12345  
addi   x6, x6, 0x678
```

Data Processing Instructions Cont'd

- Generating 32-bit constants

```
// Generating 32-bit constant  
y = 0x12345678;
```

```
// Assume x6 = y  
lui    x6, 0x12345  
addi   x6, x6, 0x678
```

```
// Generating 32-bit constant  
y = 0x12345A78;
```

```
// Assume x6 = y  
lui    x6, 0x12346  
addi   x6, x6, 0xFFFFA78
```

Data Transfer Instructions

- Example data transfer instructions and their descriptions

```
lw    x3, 0x23(x5)      // x3 = Mem[x5 + 0x23][31:0]

// loads byte/halfword and sign extends it
lb     x2, 0x34(x4)      // x2 = Mem[x4 + 0x34][7:0]
lh     x2, 0x34(x4)      // x2 = Mem[x4 + 0x34][15:0]

// loads halfword and zero extends it
lhu    x2, 0x34(x4)      // x2 = Mem[x4 + 0x34][15:0]

sh     x8, 0x45(x7)      // Mem[x7 + 0x45][15:0] = x8[15:0]
```

Data Transfer Instructions

- Example data transfer instructions and their descriptions

```
lw    x3, 0x23(x5)      // x3 = Mem[x5 + 0x23][31:0]

// loads byte/halfword and sign extends it
lb     x2, 0x34(x4)      // x2 = Mem[x4 + 0x34][7:0]
lh     x2, 0x34(x4)      // x2 = Mem[x4 + 0x34][15:0]

// loads halfword and zero extends it
lhu    x2, 0x34(x4)      // x2 = Mem[x4 + 0x34][15:0]

sh     x8, 0x45(x7)      // Mem[x7 + 0x45][15:0] = x8[15:0]
```

- Loading a global variable (beyond 12-bit offset range)

```
auipc  x7, 23456         // rd = mem[PC + 0x23456789]
lw     x9, 789(x7)
```

Flow Control Instructions

- Unconditional (jump) and conditional (branch) instructions and their descriptions

```
beq    rs1, rs2, Label    // if(rs1 == rs2) {PC = PC + imm}
blt    rs1, rs2, Label    // if(rs1 < rs2) {PC = PC + imm}

// if x10 >= x11 or x10 < 0, then goto OutOfBound
bgeu   x10, x11, OutOfBound

jal     x1, Label         // x1 = PC + 4; PC = PC + imm
jalr    x1, imm(rs1)      // x1 = PC + 4; PC = rs1 + imm
```

```
ecall                                // Transfer control to OS
ebreak                               // Transfer control to debugger
```


Flow Control Instructions Cont'd

- Calling a nearby (within 20-bit offset range for JAL instruction) function

```
jal    ra, Label    // ra = PC + 4; PC = PC + imm = Label
```

Flow Control Instructions Cont'd

- Calling a nearby (within 20-bit offset range for JAL instruction) function

```
jal    ra, Label    // ra = PC + 4; PC = PC + imm = Label
```

- Calling a far away (beyond 20-bit offset range) function
- `call` is the corresponding pseudo-assembly instruction

```
auipc  ra, 0x12345    // offset = Label - PC = 0x12345678  
jalr   ra, 0x678(ra)  // ra = PC + 4; PC = ra + 0x678
```

Assembly Programming: if-else Construct

- **if-else** in C and assembly
- Assume $f \sim x10$, $g \sim x11$, $h \sim x12$, $i \sim x13$, $j \sim x14$

```
if (i == j)
    f = g + h;
else
    f = g - h;
```

Listing 1: C code.

```
bne    x13, x14, label    # if (i == j)
add    x10, x11, x12      # f = g + h;
j      exit

label:
sub    x10, x11, x12      # f = g - h;
exit:
```

Listing 2: Assembly code for if-else.

Assembly Programming: while Loop

- GCD algorithm in C and assembly
- Assume $a \sim x8$ and $b \sim x9$

```
// GCD implementation based on
// Euclid algorithm
int gcd(int a, int b)
{
    while (a != b)
    {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}
```

Listing 3: C code for GCD.

```
addi    x8, x0, 12
addi    x9, x0, 9

gcd:
    beq    x8, x9, stop
    blt    x8, x9, less
    sub    x8, x8, x9
    j      gcd
less:
    sub    x9, x9, x8
    j      gcd
stop:
    j      stop
```

Listing 4: GCD assembly code.

Pseudo Assembly Instructions

Table 1: Selected pseudo assembly instructions.

Pseudo Instruction	Base	Instruction	Description
nop	addi	x0, x0, 0	No operation
neg rd, rs	sub	rd, x0, 0	Two's complement
j offset	jal	x0, offset	Jump
mv rd, rs	addi	rd, rs, 0	Copy register
not rd, rs	xori	rd, rs, -1	One's complement
li rd, imm	lui	x9, imm	Load immediate uses
	addi	rd, x9, imm	lui and addi
ret	jalr	x0, ra, 0	return from function
call offset	auipc	x1, offset[31:12]	Call uses
	jalr	x1, x1, offset[11:0]	auipc and jalr

Function Call Conventions

- Function parameters and return values conventions
 - Put parameters in a place where the function being called (**callee**) can access them
 - Transfer control to the callee
 - Acquire (local) resources required by the callee
 - Perform desired task of the callee
 - Put result(s) in a place where **caller** (point of origin) can access it
 - Return control to caller

Function Call Conventions Cont'd

- Registers for parameter passing
 - Registers are faster than memory, use them for parameter passing and return values
 - a0–a7 (x10–x17): eight registers for passing parameters
 - a0–a1 (x10–x11): two registers for return values
 - ra (x1): return address register to return to point of origin

Function Call Conventions Cont'd

- Registers for parameter passing
 - Registers are faster than memory, use them for parameter passing and return values
 - a0–a7 (x10–x17): eight registers for passing parameters
 - a0–a1 (x10–x11): two registers for return values
 - ra (x1): return address register to return to point of origin
- Registers preserved across function calls
 - sp, gp, tp, s0–s11 (saved registers, s0 is also frame pointer (fp))

Function Call Conventions Cont'd

- Registers for parameter passing
 - Registers are faster than memory, use them for parameter passing and return values
 - a0–a7 (x10–x17): eight registers for passing parameters
 - a0–a1 (x10–x11): two registers for return values
 - ra (x1): return address register to return to point of origin
- Registers preserved across function calls
 - sp, gp, tp, s0–s11 (saved registers, s0 is also frame pointer (fp))
- Registers that are not preserved across function calls
 - a0–a7 (argument and return registers), ra, t0–t6 (temporary registers)

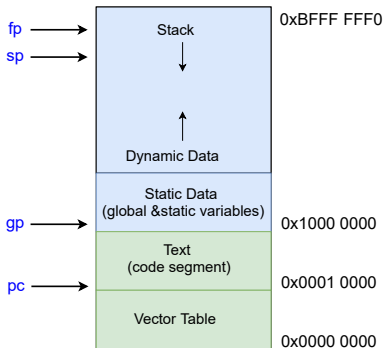
Function Call Conventions Cont'd

Table 2: Assembler mnemonics and register conventions for RV32I.

Register	ABI Name	Description	Saver
x0	zero	Hardwired to 0	—
x1	ra	Return address for subroutine calls	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global Pointer	—
x4	tp	Thread pointer	—
x5-x7	t0-t2	Memory temporary registers	Caller
x8	s0/fp	Frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Arguments to subroutines/return values	Caller
x12-x17	a2-a7	Arguments to subroutines	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporary registers	Caller

Memory Allocation

- An example memory allocation for program and data



Caller-Callee Working Example

```
int add(int x, int y) {  
    return x+y;  
}  
  
int main(void) {  
    // declare some variables  
    int x = 123, y = 987, z = 0;  
  
    // call the user function  
    z = add(x,y);  
  
    // endless loop  
    while(1){}  
}
```

Caller-Callee Working Example Cont'd

1 - Stack pointer is adjusted

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw    s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw    a0,-20(s0)
9c: feb42423      sw    a1,-24(s0)
a0: fec42703      lw    a4,-20(s0)
a4: fe842783      lw    a5,-24(s0)
a8: 00f707b3      add   a5,a4,a5
ac: 00078513      mv    a0,a5
b0: 01c12403      lw    s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw    ra,28(sp)
c4: 00812c23      sw    s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li    a5,123
d0: fef42623      sw    a5,-20(s0)
d4: 3db00793      li    a5,987
d8: fef42423      sw    a5,-24(s0)
dc: fe042223      sw    zero,-28(s0)
e0: fe842583      lw    a1,-24(s0)
e4: fec42503      lw    a0,-20(s0)
e8: fa5ff0ef      jal   ra,8c <add>
ec: fea42223      sw    a0,-28(s0)
f0: 0000006f      j     f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

2 - Save old frame pointer
(fp/s0)

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw    s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw    a0,-20(s0)
9c: feb42423      sw    a1,-24(s0)
a0: fec42703      lw    a4,-20(s0)
a4: fe842783      lw    a5,-24(s0)
a8: 00f707b3      add   a5,a4,a5
ac: 00078513      mv    a0,a5
b0: 01c12403      lw    s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw    ra,28(sp)
c4: 00812c23      sw    s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li    a5,123
d0: fef42623      sw    a5,-20(s0)
d4: 3db00793      li    a5,987
d8: fef42423      sw    a5,-24(s0)
dc: fe042223      sw    zero,-28(s0)
e0: fe842583      lw    a1,-24(s0)
e4: fec42503      lw    a0,-20(s0)
e8: fa5ff0ef      jal   ra,8c <add>
ec: fea42223      sw    a0,-28(s0)
f0: 0000006f      j     f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

3 - Frame pointer (fp/s0) points to the start of stack frame

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw   s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw   a0,-20(s0)
9c: feb42423      sw   a1,-24(s0)
a0: fec42703      lw   a4,-20(s0)
a4: fe842783      lw   a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv   a0,a5
b0: 01c12403      lw   s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw   ra,28(sp)
c4: 00812c23      sw   s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li   a5,123
d0: fef42623      sw   a5,-20(s0)
d4: 3db00793      li   a5,987
d8: fef42423      sw   a5,-24(s0)
dc: fe042223      sw   zero,-28(s0)
e0: fe842583      lw   a1,-24(s0)
e4: fec42503      lw   a0,-20(s0)
e8: fa5ff0ef      jal  ra,8c <add>
ec: fea42223      sw   a0,-28(s0)
f0: 0000006f      j    f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

4 - Save the function arguments

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw    s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw    a0,-20(s0)
9c: feb42423      sw    a1,-24(s0)
a0: fec42703      lw    a4,-20(s0)
a4: fe842783      lw    a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv    a0,a5
b0: 01c12403      lw    s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw    ra,28(sp)
c4: 00812c23      sw    s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li    a5,123
d0: fef42623      sw    a5,-20(s0)
d4: 3db00793      li    a5,987
d8: fef42423      sw    a5,-24(s0)
dc: fe042223      sw    zero,-28(s0)
e0: fe842583      lw    a1,-24(s0)
e4: fec42503      lw    a0,-20(s0)
e8: fa5ff0ef      jal   ra,8c <add>
ec: fea42223      sw    a0,-28(s0)
f0: 0000006f      j     f0 <main+0x34>
```


Caller-Callee Working Example Cont'd

5 - Make a local copy of the function arguments

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw  s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw  a0,-20(s0)
9c: feb42423      sw  a1,-24(s0)
a0: fec42703      lw  a4,-20(s0)
a4: fe842783      lw  a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv   a0,a5
b0: 01c12403      lw  s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw  ra,28(sp)
c4: 00812c23      sw  s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li   a5,123
d0: fef42623      sw  a5,-20(s0)
d4: 3db00793      li   a5,987
d8: fef42423      sw  a5,-24(s0)
dc: fe042223      sw  zero,-28(s0)
e0: fe842583      lw  a1,-24(s0)
e4: fec42503      lw  a0,-20(s0)
e8: fa5ff0ef      jal  ra,8c <add>
ec: fea42223      sw  a0,-28(s0)
f0: 0000006f      j    f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

6 - Perform actual operation

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw  s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw  a0,-20(s0)
9c: feb42423      sw  a1,-24(s0)
a0: fec42703      lw  a4,-20(s0)
a4: fe842783      lw  a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv  a0,a5
b0: 01c12403      lw  s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw  ra,28(sp)
c4: 00812c23      sw  s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li  a5,123
d0: fef42623      sw  a5,-20(s0)
d4: 3db00793      li  a5,987
d8: fef42423      sw  a5,-24(s0)
dc: fe042223      sw  zero,-28(s0)
e0: fe842583      lw  a1,-24(s0)
e4: fec42503      lw  a0,-20(s0)
e8: fa5ff0ef      jal ra,8c <add>
ec: fea42223      sw  a0,-28(s0)
f0: 0000006f      j   f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

7 - Return the result

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
00000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw  s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw  a0,-20(s0)
9c: feb42423      sw  a1,-24(s0)
a0: fec42703      lw  a4,-20(s0)
a4: fe842783      lw  a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv   a0,a5
b0: 01c12403      lw  s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
0000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw  ra,28(sp)
c4: 00812c23      sw  s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li  a5,123
d0: fef42623      sw  a5,-20(s0)
d4: 3db00793      li  a5,987
d8: fef42423      sw  a5,-24(s0)
dc: fe042223      sw  zero,-28(s0)
e0: fe842583      lw  a1,-24(s0)
e4: fec42503      lw  a0,-20(s0)
e8: fa5ff0ef      jal ra,8c <add>
ec: fea42223      sw  a0,-28(s0)
f0: 0000006f      j   f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

8 - Revert the frame pointer (s0)
and stack pointer (sp)

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {

    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw  s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw  a0,-20(s0)
9c: feb42423      sw  a1,-24(s0)
a0: fec42703      lw  a4,-20(s0)
a4: fe842783      lw  a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv   a0,a5
b0: 01c12403      lw  s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw  ra,28(sp)
c4: 00812c23      sw  s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li   a5,123
d0: fef42623      sw  a5,-20(s0)
d4: 3db00793      li   a5,987
d8: fef42423      sw  a5,-24(s0)
dc: fe042223      sw  zero,-28(s0)
e0: fe842583      lw  a1,-24(s0)
e4: fec42503      lw  a0,-20(s0)
e8: fa5ff0ef      jal  ra,8c <add>
ec: fea42223      sw  a0,-28(s0)
f0: 0000006f      j    f0 <main+0x34>
```

Caller-Callee Working Example Cont'd

9 - Return from the function
(jalr x0, ra, 0)

```
int add(int x, int y) {
    return x+y;
}

int main(void) {
    // declare some variables
    int x = 123, y = 987, z = 0;

    // call the user function
    z = add(x, y);

    // endless loop
    while(1) {
    }
}
```

```
0000008c <add>:
8c: fe010113      addi sp,sp,-32
90: 00812e23      sw  s0,28(sp)
94: 02010413      addi s0,sp,32
98: fea42623      sw  a0,-20(s0)
9c: feb42423      sw  a1,-24(s0)
a0: fec42703      lw  a4,-20(s0)
a4: fe842783      lw  a5,-24(s0)
a8: 00f707b3      add  a5,a4,a5
ac: 00078513      mv   a0,a5
b0: 01c12403      lw  s0,28(sp)
b4: 02010113      addi sp,sp,32
b8: 00008067      ret
```

```
000000bc <main>:
bc: fe010113      addi sp,sp,-32
c0: 00112e23      sw  ra,28(sp)
c4: 00812c23      sw  s0,24(sp)
c8: 02010413      addi s0,sp,32
cc: 07b00793      li   a5,123
d0: fef42623      sw  a5,-20(s0)
d4: 3db00793      li   a5,987
d8: fef42423      sw  a5,-24(s0)
dc: fe042223      sw  zero,-28(s0)
e0: fe842583      lw  a1,-24(s0)
e4: fec42503      lw  a0,-20(s0)
e8: fa5ff0ef      jal  ra,8c <add>
ec: fea42223      sw  a0,-28(s0)
f0: 0000006f      j    f0 <main+0x34>
```

Suggested Reading

- Read Chapter 2 of [[Patterson and Hennessy, 2021](#)].
- Read User Manual for the instruction set and its architecture [[Waterman et al., 2016b](#)].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References



Patterson, D. and Hennessy, J. (2021).

Computer Organization and Design RISC-V Edition: The Hardware Software Interface, 2nd Edition.

Morgan Kaufmann.



Waterman, A., Lee, Y., Avizienis, R., Patterson, D. A., and Asanovic, K. (2016a).

The risc-v instruction set manual volume ii: Privileged architecture version 1.9.

EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.



Waterman, A., Lee, Y., Patterson, D. A., and Asanović, K. (2016b).

The risc-v instruction set manual, volume i: User-level isa, version 2.1.

EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129.