AI ASIC: Design and Practice (ADaP)

Fall 2024

Two Examples for AI Chip

燕博南

Two Examples

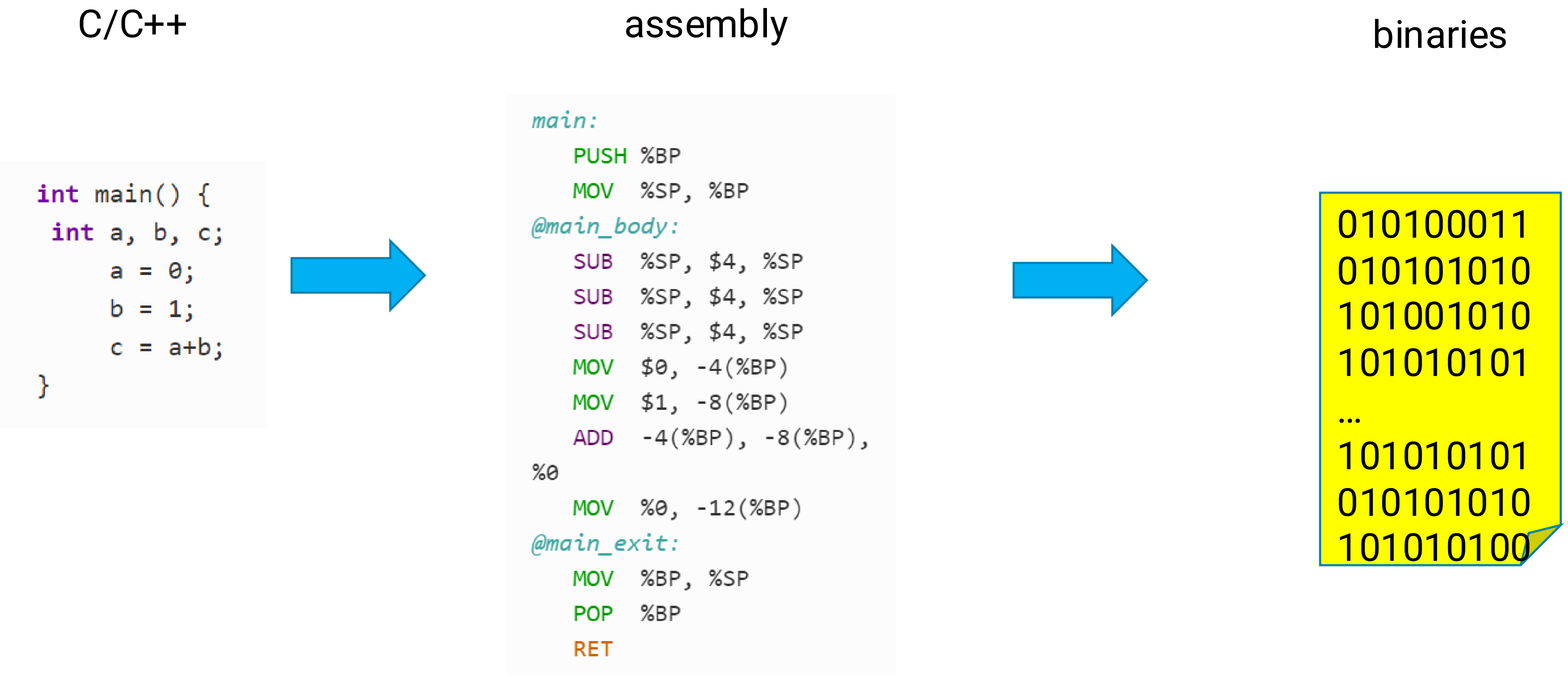
1. Closely-Coupled Accelerators
2. Loosely-Coupled Accelerators

Two Examples

1. **Closely-Coupled Accelerators**
2. Loosely-Coupled Accelerators

Hardware/Software Interface

Compile



C/C++

```
int main() {
 int a, b, c;
     a = 0;
     b = 1;
     c = a+b;
}
```

assembly

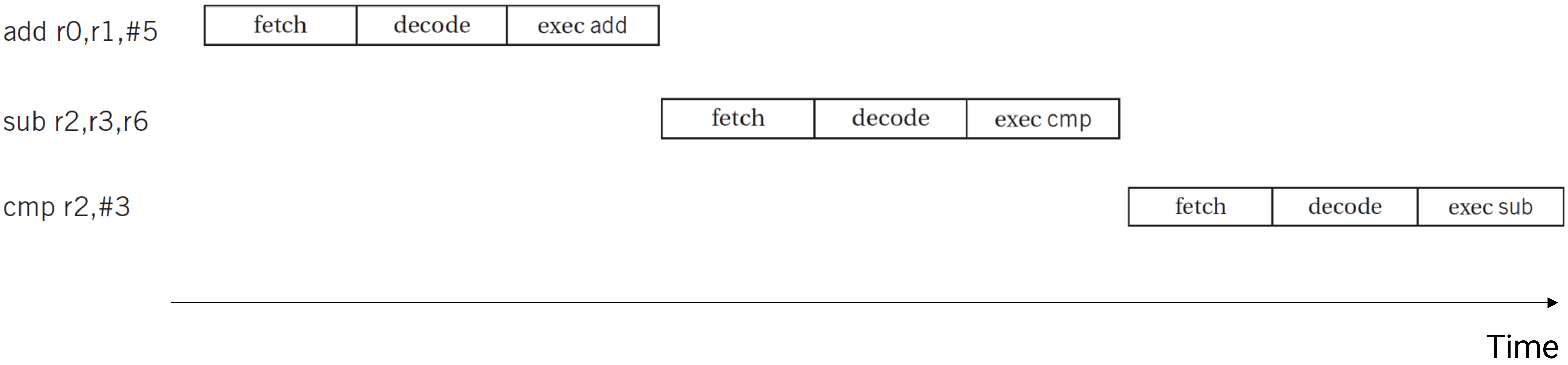
```
main:
   PUSH %BP
   MOV  %SP, %BP
@main_body:
   SUB  %SP, $4, %SP
   SUB  %SP, $4, %SP
   SUB  %SP, $4, %SP
   MOV  $0, -4(%BP)
   MOV  $1, -8(%BP)
   ADD  -4(%BP), -8(%BP),
%0
   MOV  %0, -12(%BP)
@main_exit:
   MOV  %BP, %SP
   POP  %BP
   RET
```

binaries

```
010100011
010101010
101001010
101010101
…
101010101
010101010
101010100
```

Pipelined Execution

Single-Stage Execution



add r0,r1,#5

fetch | decode | exec add

sub r2,r3,r6

fetch | decode | exec cmp

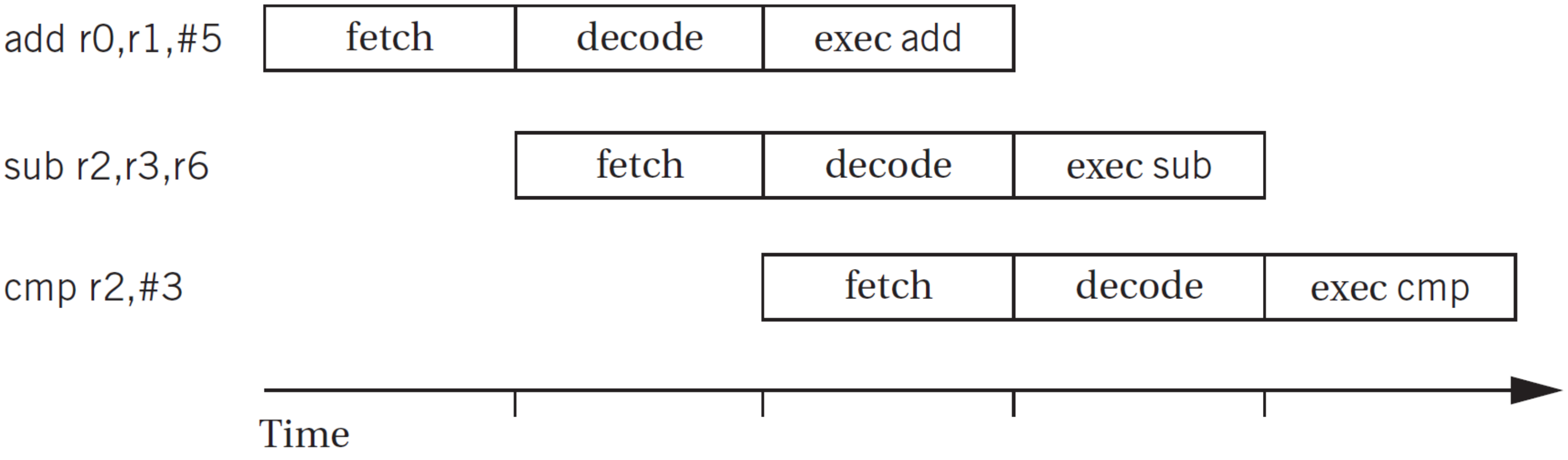
cmp r2,#3

fetch | decode | exec sub

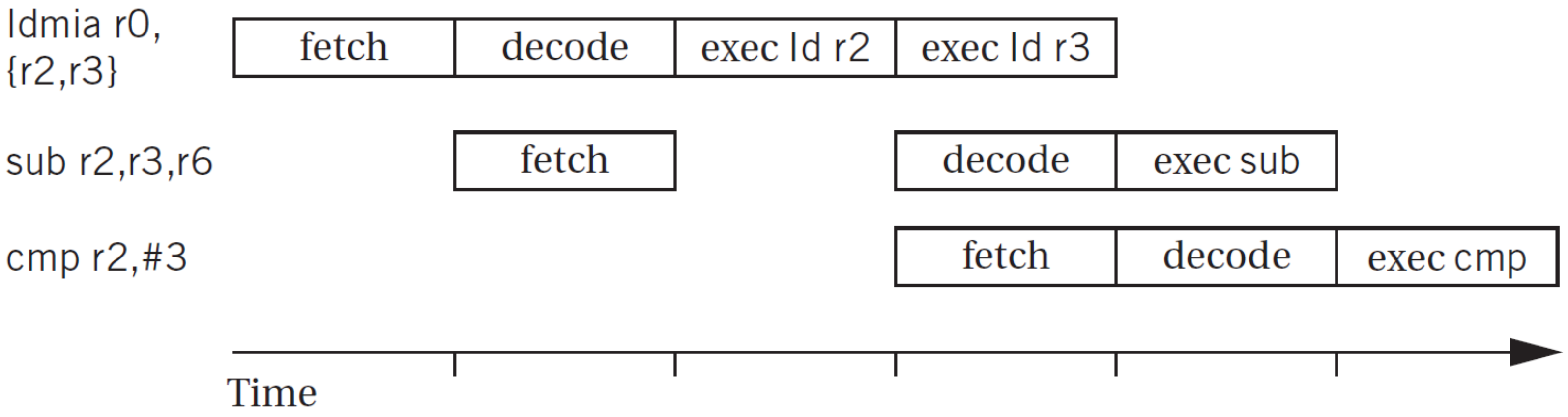
Time

Pipeline

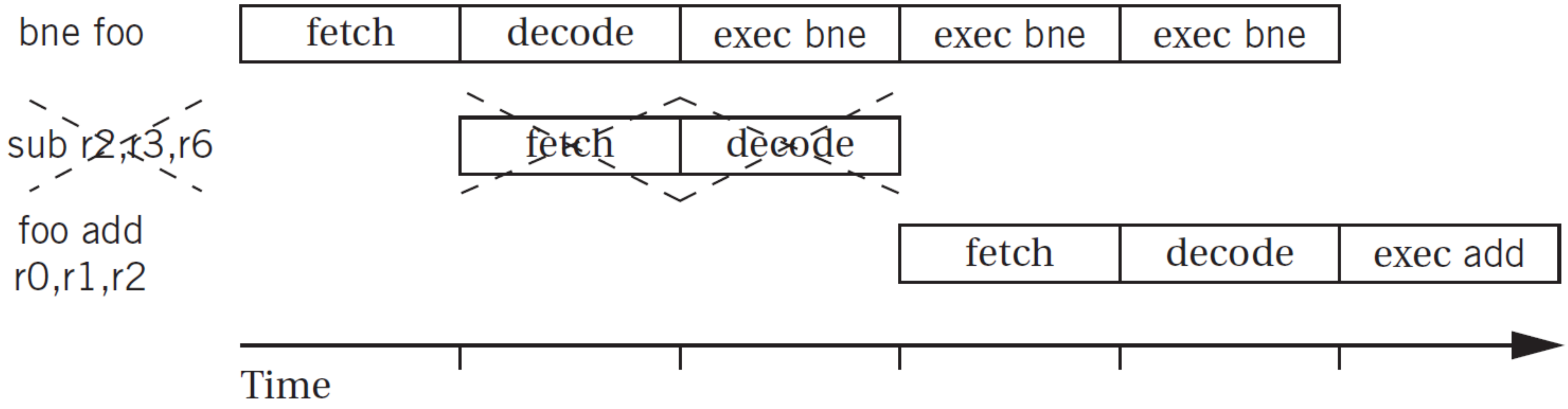
Single-Cycle Instruction



Multi-Cycle Instruction

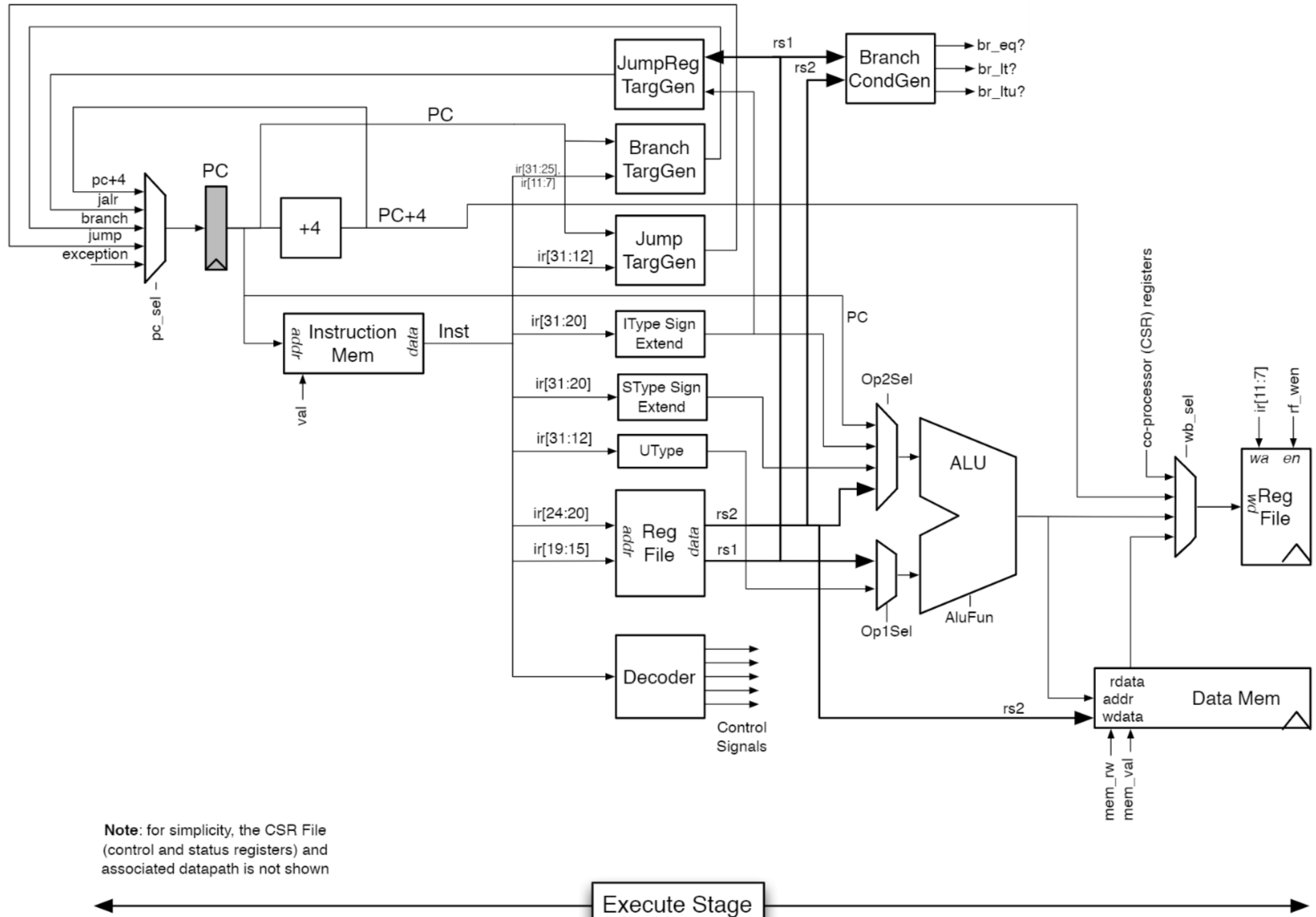


Pipeline A Branch?



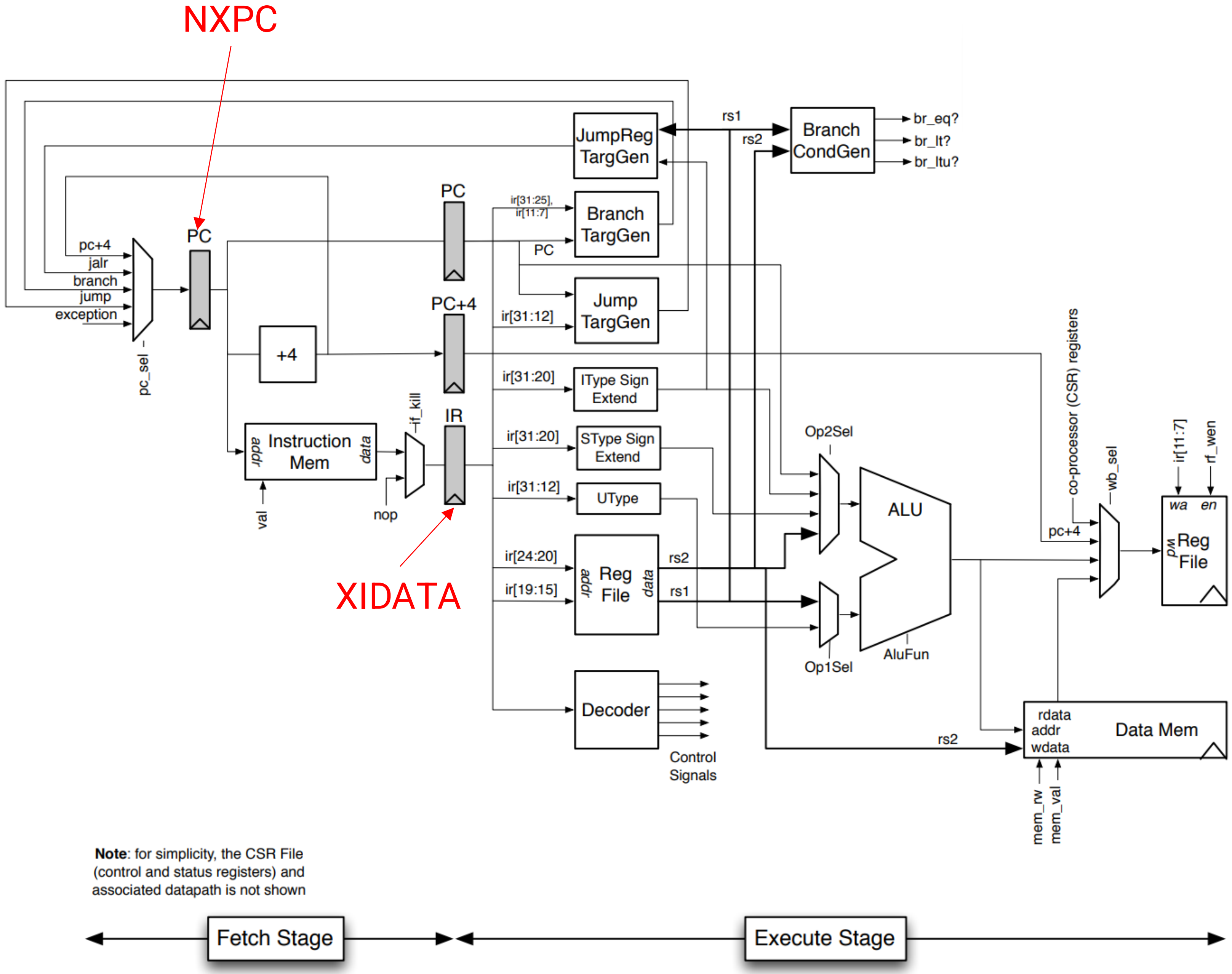
Use “flush” to retry obtaining next PC that should point to

Hardware View of Single-Stage CPU



Note: for simplicity, the CSR File (control and status registers) and associated datapath is not shown

Hardware View of 2-Stage CPU



Note: for simplicity, the CSR File (control and status registers) and associated datapath is not shown

Let’s see code directly

- Revised from DarkRISCV
- Remove “3-stage”, “hardware threads”, “flex bit-width” features

Hints

Steps:

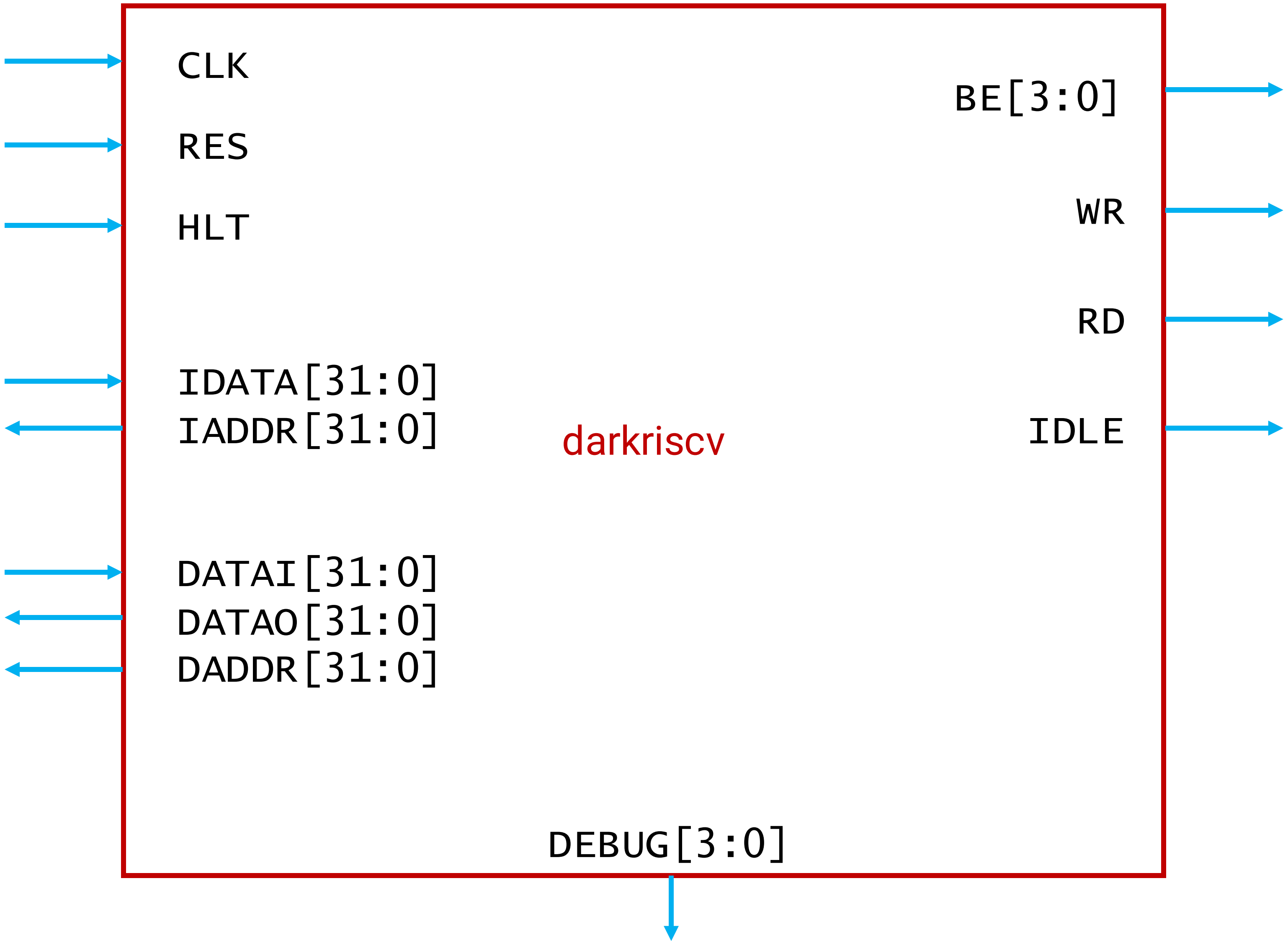
1) Look at the ports
2) Get familiar with RISC-V assembly, especially RV32I base
3) Read 2 “always” block
4) Check pipeline

Hints

Steps:

1) Look at the ports
2) Get familiar with RISC-V assembly, especially RV32I base
3) Read 2 “always” block
4) Check pipeline

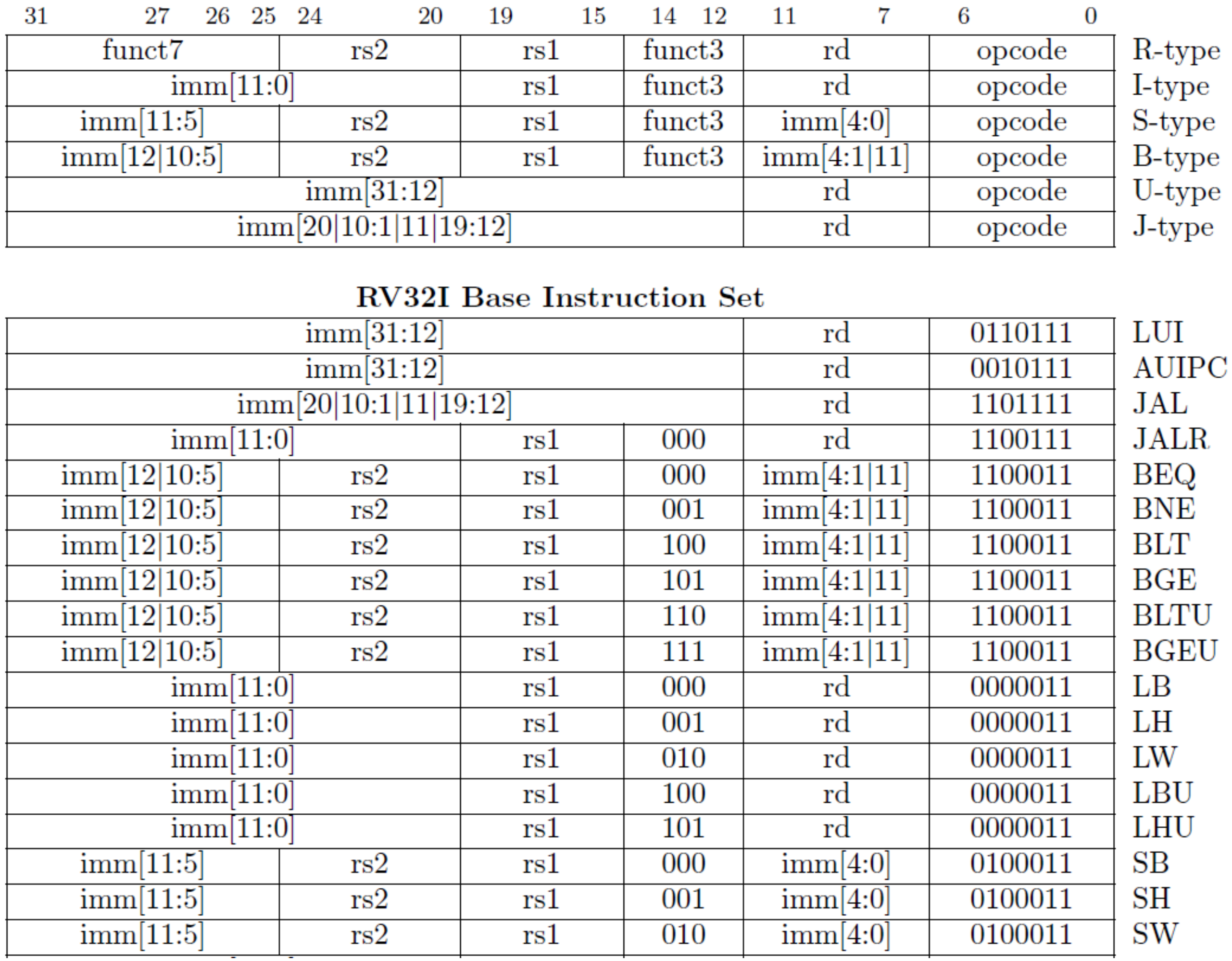
Top Module


CLK
RES
HLT
IDATA[31:0]
IADDR[31:0]
DATAI[31:0]
DATAO[31:0]
DADDR[31:0]
darkriscv
BE[3:0]
WR
RD
IDLE
DEBUG[3:0]

Hints

Steps:

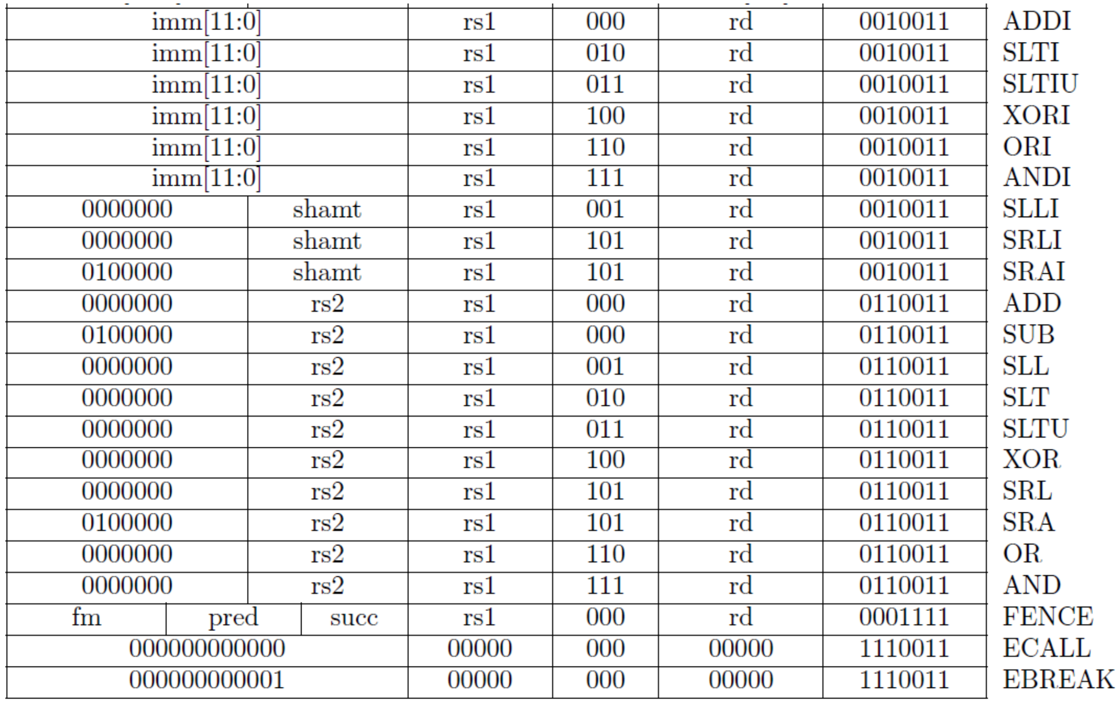
1) Look at the ports
2) Get familiar with RISC-V assembly, especially RV32I base
3) Read 2 “always” block
4) Check pipeline



31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

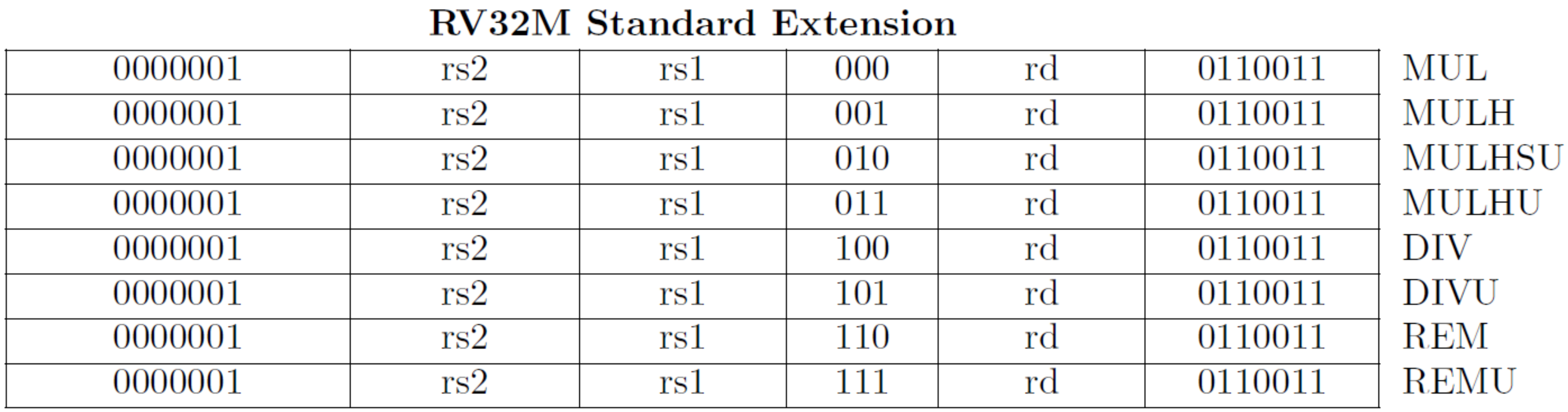
RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW



imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

M Extension



RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

Hints

Steps:

1) Look at the ports
2) Get familiar with RISC-V assembly, especially RV32I base
3) Read 2 “always” block
4) Check pipeline

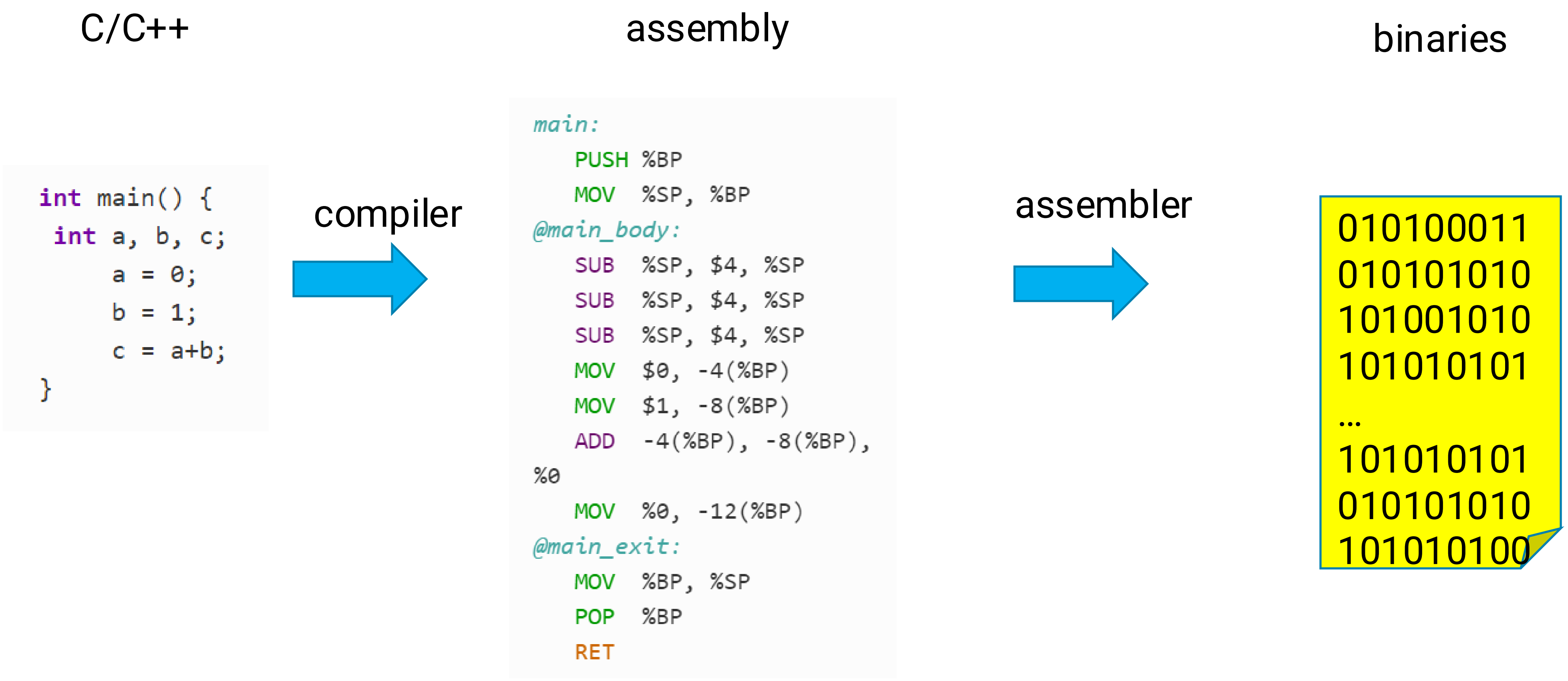
Hints

Steps:

1) Look at the ports
2) Get familiar with RISC-V assembly, especially RV32I base
3) Read 2 “always” block
4) Check pipeline

How to make a simple testbench for CPUs?

Compile & Assemble



C/C++

```c
int main() {
 int a, b, c;
      a = 0;
      b = 1;
      c = a+b;
}
```

compiler

assembly

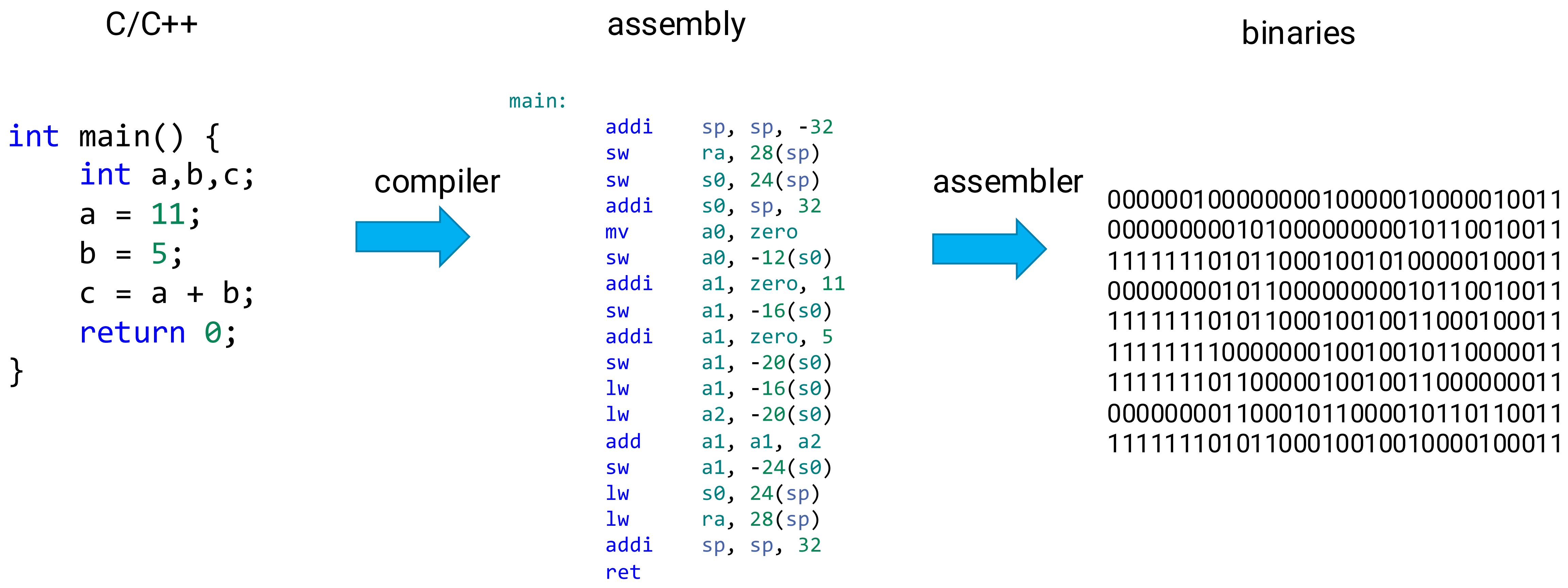
```
main:
   PUSH %BP
   MOV  %SP, %BP
@main_body:
   SUB  %SP, $4, %SP
   SUB  %SP, $4, %SP
   SUB  %SP, $4, %SP
   MOV  $0, -4(%BP)
   MOV  $1, -8(%BP)
   ADD  -4(%BP), -8(%BP),
%0
   MOV  %0, -12(%BP)
@main_exit:
   MOV  %BP, %SP
   POP  %BP
   RET
```

assembler

binaries

```
010100011
010101010
101001010
101010101
…
101010101
010101010
101010100
```

Compile & Assemble



C/C++

```
int main() {
    int a,b,c;
    a = 11;
    b = 5;
    c = a + b;
    return 0;
}
```

compiler

assembly

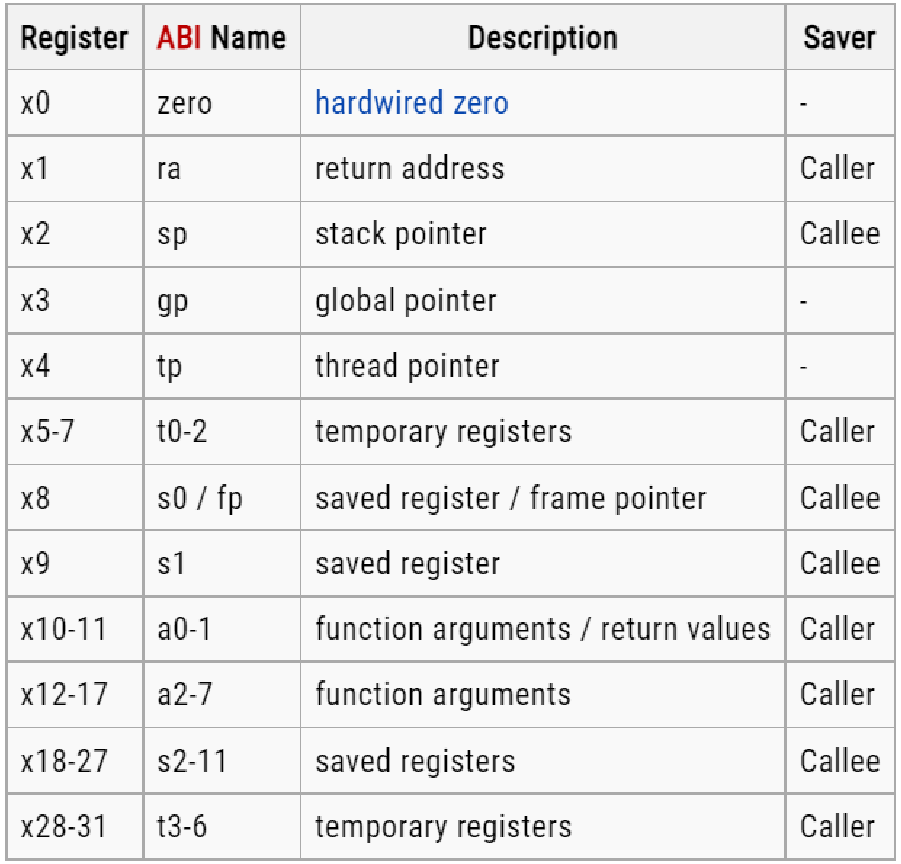
```
main:
        addi    sp, sp, -32
        sw      ra, 28(sp)
        sw      s0, 24(sp)
        addi    s0, sp, 32
        mv      a0, zero
        sw      a0, -12(s0)
        addi    a1, zero, 11
        sw      a1, -16(s0)
        addi    a1, zero, 5
        sw      a1, -20(s0)
        lw      a1, -16(s0)
        lw      a2, -20(s0)
        add     a1, a1, a2
        sw      a1, -24(s0)
        lw      s0, 24(sp)
        lw      ra, 28(sp)
        addi    sp, sp, 32
        ret
```

assembler

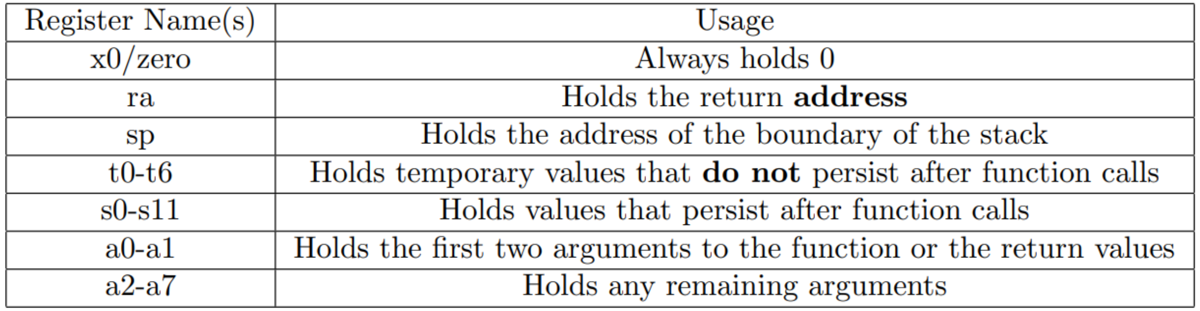
binaries

```
00000010000000010000010000010011
00000000010100000000010110010011
11111110101100010010100000100011
00000000101100000000010110010011
11111110101100010010011000100011
11111111000000001001001011000011
11111110110000010010011000000011
00000000110001011000010110110011
11111110101100010010010000100011
```

Registers

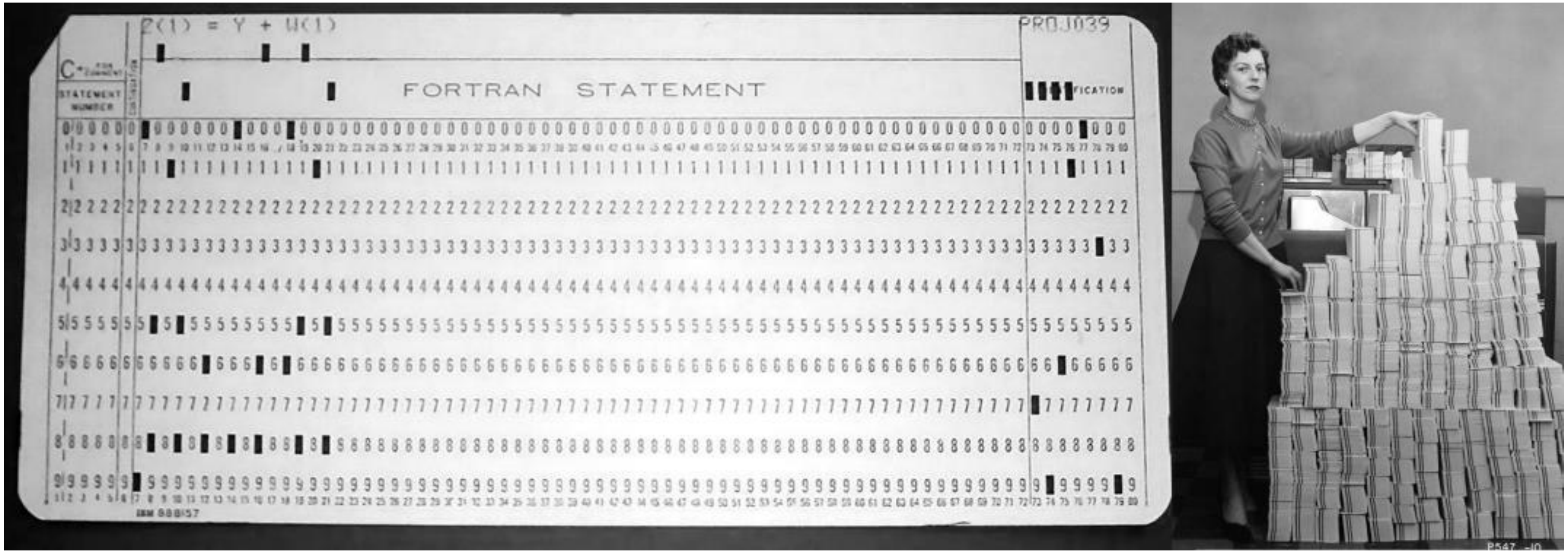


Register	ABI Name	Description	Saver
x0	zero	hardwired zero	-
x1	ra	return address	Caller
x2	sp	stack pointer	Callee
x3	gp	global pointer	-
x4	tp	thread pointer	-
x5-7	t0-2	temporary registers	Caller
x8	s0 / fp	saved register / frame pointer	Callee
x9	s1	saved register	Callee
x10-11	a0-1	function arguments / return values	Caller
x12-17	a2-7	function arguments	Caller
x18-27	s2-11	saved registers	Callee
x28-31	t3-6	temporary registers	Caller



Register Name(s)	Usage
x0/zero	Always holds 0
ra	Holds the return **address**
sp	Holds the address of the boundary of the stack
t0-t6	Holds temporary values that **do not** persist after function calls
s0-s11	Holds values that persist after function calls
a0-a1	Holds the first two arguments to the function or the return values
a2-a7	Holds any remaining arguments

Historic View



Some Useful Toolkits

- Compiler Explorer (godbolt.org)
- RISC-V Interpreter (cornell.edu)
- riscv-assembler (riscvassembler.org)

Two Examples

1. Closely-Coupled Accelerators
2. **Loosely-Coupled Accelerators**
    - **PUMA (equip specialized MAC with programmability)**