CO Midterm Cheatsheet

指令集相关

Category	Inst	Example & Comments			FMT	OpCode	Funct3	Funct6/7
Arithmetic	add	Add	add rd, rs1, rs2	rd = rs1 + rs2	R	0110011	000	0000000
	sub	Subtract	sub rd, rs1, rs2	rd = rs1 - rs2	R	0110011	000	0100000
	addi	Add imm	addi rd, rs1, -20	rd = rs1 + (-20)	I	0010011	000	n.a.
	slt	Set if less than	slt rd, rs1, rs2	rd = rs1 < rs2 ? 1 : 0	R	0110011	010	0000000
	sltu	slt, unsigned	sltu rd, rs1, rs2	rd = rs1 < rs2 ? 1 : 0	R	0110011	011	0000000
	slti	slt, imm	slti rd, rs1, imm	rd = rs1 < imm ? 1 : 0	I	0010011	010	n.a.
	sltiu	slt, imm & unsigned	sltiu rd, rs1, imm	rd = rs1 < imm ? 1 : 0	I	0010011	011	n.a.
	mul	mul, lower 64 of result	mul rd, rs1, rs2	rd = rs1 * rs2 (lower 64)	R	-	-	-
	mulh	mul, upper 64 of result	mulh rd, rs1, rs2	rd = (rs1 * rs2) >> 64	R	-	-	-
	mulhu	mulh, unsgn * unsgn	mulhu rd, rs1, rs2	rd = (rs1 * rs2) >> 64	R	-	-	-
	mulhsu	mulh, sgn * unsgn	mulhsu rd, rs1, rs2	rd = (rs1 * rs2) >> 64	R	-	-	-
	div	divide	div rd, rs1, rs2	rd = rs1 / rs2	R	-	-	-
	divu	divide unsigned	divu rd, rs1, rs2	rd = rs1 / rs2	R	-	-	-
	rem	remainder	rem rd, rs1, rs2	rd = rs1 % rs2	R	-	-	-
	remu	remainder unsigned	remu rd, rs1, rs2	rd = rs1 % rs2	R	-	-	-
Data transfer	ld	Load dword	ld rd, 40(rs1)	rd = [rs1 + 40]	I	0000011	011	n.a.
	sd	Store dword	sd rs2, 40(rs1)	[rs1 + 40] = rs2	S	0100011	011	n.a.
	lw	Load word	-	-	I	0000011	010	n.a.
	lwu	Load unsgn word	-	-	I	0000011	110	n.a.
	sw	Store word	-	-	S	0100011	010	n.a.
	lh	Load half word	-	-	I	0000011	001	n.a.
	lhu	Load unsgn hword	-	-	I	0000011	101	n.a.
	sh	Store hword	-	-	S	0100011	001	n.a.
	lb	Load byte	-	-	I	0000011	000	n.a.
	lbu	Load unsgn byte	-	-	I	0000011	100	n.a.
	sb	Store byte	-	-	S	0100011	000	n.a.
	lui	Load upper imm	lui rd, 0x12345	rd = 0x12345000	U	0110111	n.a.	n.a.
	auipc	Add upper imm to PC	auipc rd, 0x12345	rd = PC + 0x12345000	U	0010111	n.a.	n.a.
Logical	and	And	and rd, rs1, rs2	rd = rs1 & rs2	R	0110011	111	0000000
	or	Or	or rd, rs1, rs2	rd = rs1 | rs2	R	0110011	110	0000000
	xor	Exclusive or	xor rd, rs1, rs2	rd = rs1 ^ rs2	R	0110011	100	0000000
	andi	And imm	andi rd, rs1, imm	rd = rs1 & imm	I	0010011	111	n.a.
	ori	Or imm	ori rd, rs1, imm	rd = rs1 | imm	I	0010011	110	n.a.
	xori	Xor imm	xori, rd, rs1, imm	rd = rs1 ^ imm	I	0010011	100	n.a.
Shift	sll	shift left logical	sll rd, rs1, rs2	rd = rs1 << rs2	R	0110011	001	0000000
	srl	shift right logical	srl rd, rs1, rs2	rd = rs1 >> rs2 (zExt)	R	0110011	101	0000000
	sra	shr arithmetic	sra rd, rs1, rs2	rd = rs1 >> rs2 (sExt)	R	0110011	101	0100000
	slli	shl logical imm	slli rd, rs1, imm	rd = rs1 << imm	I	0010011	001	000000
	srli	shr logical imm	srli rd, rs1, imm	rd = rs1 >> imm (zExt)	I	0010011	101	000000
	srai	shr arith imm	srai rd, rs1, imm	rd = rs1 >> imm (sExt)	I	0010011	101	010000
Conditional Branch	beq	branch if equal	beq rs1, rs2, offset	if (rs1==rs2) PC+=offset	SB	1100011	000	n.a.
	bne	branch if not equal	bne rs1, rs2, offset	if (rs1!=rs2) PC+=offset	SB	1100011	001	n.a.
	blt	br if less than	blt rs1, rs2, offset	if (rs1<rs2) PC+=offset	SB	1100011	100	n.a.
	bge	br if greater or eq	bge rs1, rs2, offset	if (rs1>=rs2) PC+=offset	SB	1100011	101	n.a.
	bltu	blt, unsigned	bltu rs1, rs2, offset	if (rs1<rs2) PC+=offset	SB	1100011	110	n.a.
	bgeu	bge, unsigned	bgeu rs1, rs2, offset	if (rs1>=rs2) PC+=offset	SB	1100011	111	n.a.
Unconditional Branch	jal	jump and link	jal rd, offset	rd=PC+4; PC+=offset	UJ	1101111	n.a.	n.a.
	jalr	jump and link reg	jalr rd, 100(rs1)	rd=PC+4; PC=rs1+100	I	1100111	000	n.a.

References/CO-Insts.webp

	6	1	5	5	3	5	7
R	funct7		rs2	rs1	funct3	rd	opcode
I	i[11:0]			rs1	funct3	rd	opcode
I	funct6	i[5:0]		rs1	funct3	rd	opcode
S	i[11:5]		rs2	rs1	funct3	i[4:0]	opcode
SB	i[12, 10:5]		rs2	rs1	funct3	i[4:1, 11]	opcode
UJ	i[20, 10:1, 11, 19:12]					rd	opcode
U	i[31:12]					rd	opcode

References/CO-InstTypes.webp

寄存器和功能

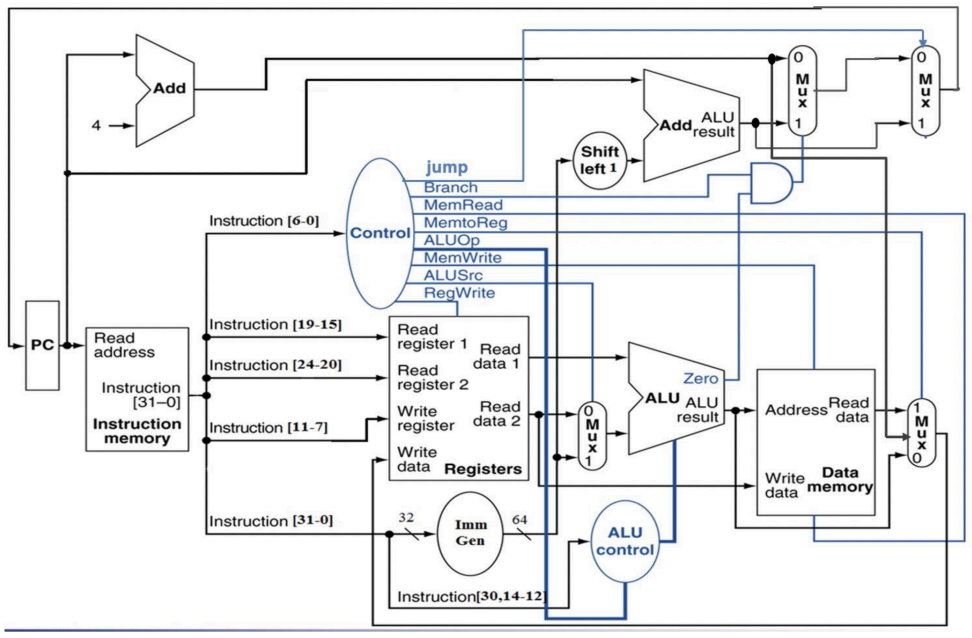
Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	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

References/CO-Regs.webp

- 保存寄存器 s
  - ！如果需要使用必须堆栈保护
- 临时寄存器 t
  - 随便用

Single Cycle CPU Structure

Truth tables & Circuitry of main Controller



输入		输出								
Instruction	OPCode	ALUSrcB	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
R-format	0110011	0	00	1	0	0	0	0	1	0
Ld(I-Type)	0000011	1	01	1	1	0	0	0	0	0
sd(S-Type)	0100011	1	X	0	0	1	0	0	0	0
beq(SB-Type)	1100111	0	X	0	0	0	1	0	0	1
Jal(UJ-Type)	1101111	X	10	1	0	0	0	1	X	X

Design the ALU Decoder second level

- **ALU operation is decided by 2-bit ALUOp derived from opcode, and funct7 & funct3 fields of the instruction**
  - Combinational logic derives ALU control

opcode	ALUOp	Operation	Funct7	funct3	ALU function	ALU control
ld	00	load register	XXXXXXX	xxx	add	0010
sd	00	store register	XXXXXXX	xxx	add	0010
beq	01	branch on equal	XXXXXXX	xxx	subtract	0110
R-type	10	add	0000000	000	add	0010
		subtract	0100000	000	subtract	0110
		AND	0000000	111	AND	0000
		OR	0000000	110	OR	0001
		SLT	0000000	010	Slt	0111

更多分析题

汇编例程

Nested Procedures

Argument n in a0
Result in a0

- **RISC-V assembly code**

```
fact:  addi   sp, sp, -16          # adjust stack for 2 items
       sd     ra, 8(sp)            # save the return address：  x1
       sd     a0, 0(sp)            # save the argument  n: x10
       addi   t0, a0, -1           # x5 = n  -  1
       bge    t0, zero, L1         # if  n  >=  1, go to L1(else)
       addi   a0, zero, 1          # return 1 if n <1
       addi   sp, sp, 16           # Recover sp (Why not recover x1and x10 ?)
       jalr   zero, 0(ra)          # return to caller
L1:    addi   a0,  a0, -1          # n  >=  1: argument gets ( n  -  1 )
       jal    ra, fact             # call fact with ( n  -  1)
       add    t1, a0, zero         #move result of fact(n - 1) to x6(t1)
       ld     a0, 0(sp)            # return from jal: restore argument n
       ld     ra, 8(sp)            # restore the return address
       add    sp, sp, 16           # adjust stack pointer to pop 2 items
       mul    a0, a0, t1           # return  n*fact ( n  -  1 )
       jalr   zero, 0(ra)          # return to the  caller
```

Misc

- `0x20241112 = 04011010422`