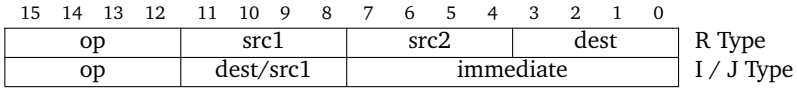
"Nah, I'd Win" 16-bit Instruction Set

Core Instruction Formats



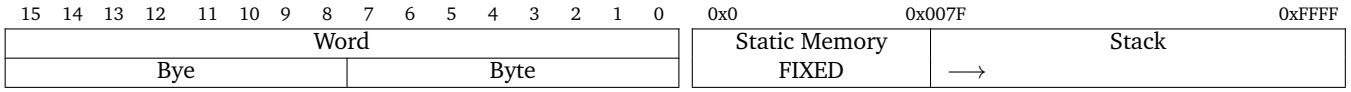
Base Integer Instructions

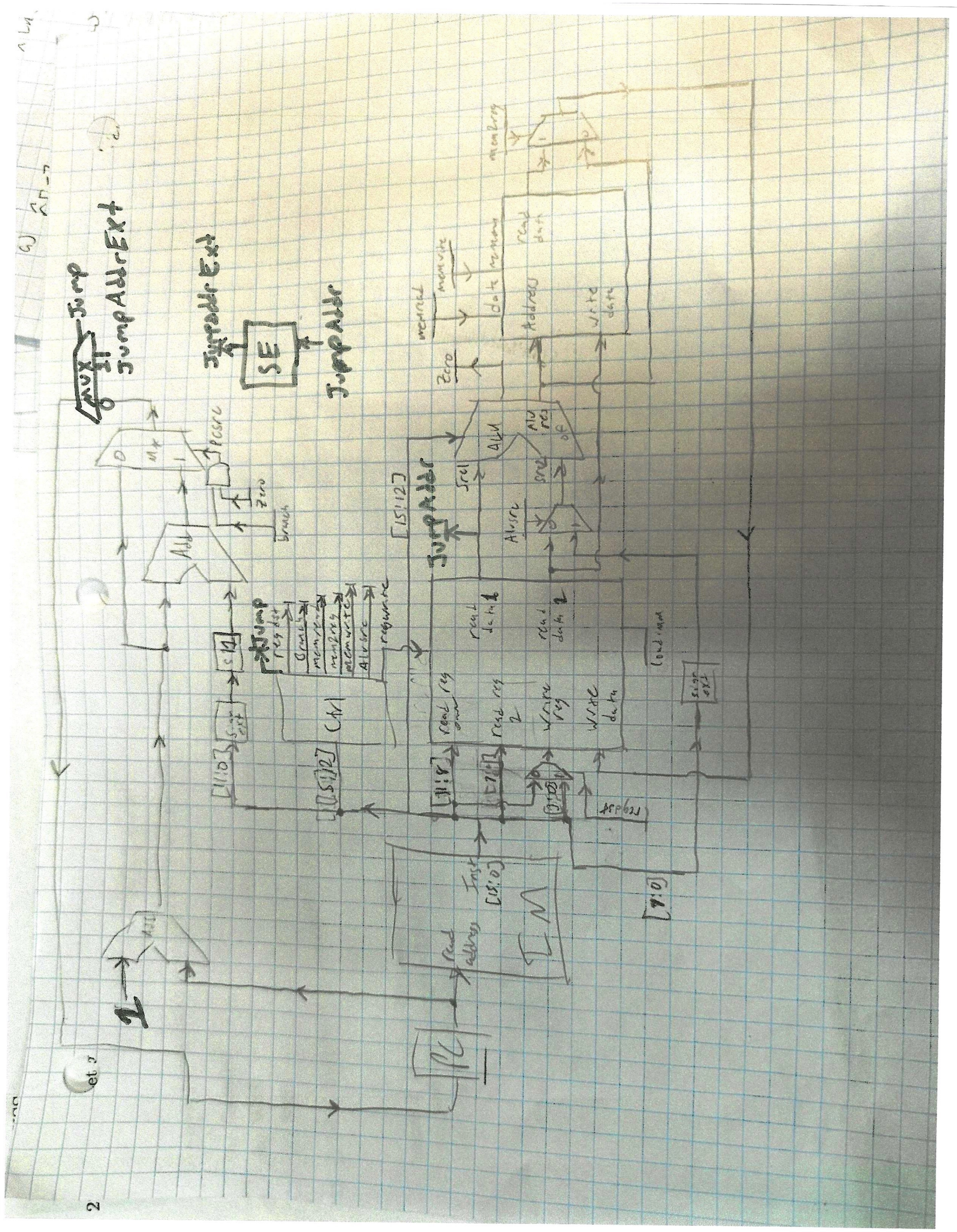
Inst	Name	FMT	Opcode	Description (C)
`NOOP`	No operation	J	`0000`	`No operation`
`JR src`	Jump register	R	`0001`	`Jump to src`
`BE src1, src2, src3`	Branch if Equal	R	`0011`	`If src1 == src2, jump to src3`
`SW src1, src2`	Store word	R	`0101`	`src1 => mem[src2]`
`LW src1, src2`	Load word	R	`0111`	`mem[src2] => src1`
`LI dest, imm`	Load immediate	I	`1000`	`dest = imm`
`ADD dest, src1, src2`	Add	R	`1001`	`dest = src1 + src2`
`XOR dest, src1, src2`	Exclusive-OR	R	`1010`	`dest = src1 ^ src2`
`AND dest, src1, src2`	AND	R	`1011`	`dest = src1 & src2`
`OR dest, src1, src2`	OR	R	`1100`	`dest = src1 | src2`
`SLL dest, src1, src2`	Shift Logical Left	R	`1110`	`dest = src1 « src2`
`SLR dest, src1, src2`	Shift Logical Right	R	`1111`	`dest = src1 » src2`

Registers

Register	ABI Name	Description	Saver
`x0`	`zero`	Zero constant	—
`x2`	`sp`	Return address	Caller
`x3`	`im`	Stack pointer	Callee
`x4`	`ra`	Immediate register	—
`x1, x5-x8`	`s,a,b,x,y`	Saved register	Callee
`x9-xd`	`t0-t5`	Temporaries	Caller
`xe-xf`	`hi, lo`	Multiplication registers	—

Data Alignment & Memory Layout (Byte-Addressed)




Jump
JumpAddrExt
SE
JumpAddr
Ctrl
ALU
PC