Number System

-ve numbers: MSB 0 for +, 1 for -

Signed & Magnitude

Range: $\pm 2^{n-1} - 1$

1s Complement

- For n-bit binary number x,$-x = 2^n - x - 1$
- Range: $\pm 2^{n-1} - 1$
- Negate all the bits

A+B:

1. Perform binary addition on the two numbers.
2. If there is a carry out of the MSB, add 1 to the result.
3. Check for overflow. Overflow occurs if result is opposite sign of A and B.

2s Complement

- For n-bit binary number x, $-x = 2^n - x$
- invert all bits and +1
- range:$-2^{n-1}\ to\ 2^{n-1} - 1$
- leftmost digit has a weight of -2^{n-1}
- **NOT TRUE** for other N's complement

A + B:

1. Perform binary addition
2. Ignore the carry out of the MSB
3. Check for overflow. Overflow occurs if the 'carry in' and 'carry out' of the MSB are different, or if result is opposite sign of A and B.

Excess Representation

- Excess-n representation to value = binary value in ex-n - n

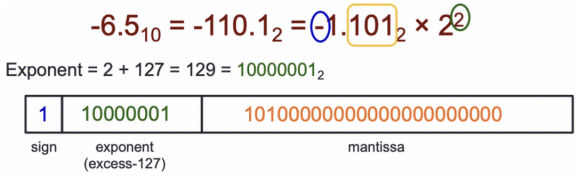
For 4-bit numbers, we may use excess-7 or excess-8 (2^4 = 16; 16 / 2 = 8)

Excess-8 Representation	*Value*
0000	-8
0001	-7
0010	-6
0011	-5
0100	-4
0101	-3
0110	-2
0111	-1

Excess-8 Representation	*Value*
1000	0
1001	1
1010	2
1011	3
1100	4
1101	5
1110	6
1111	7

Floating-Point (IEEE 754)

- Single-precision (32 bits): 1-bit sign, **8-bit exponent** with bias 127 (excess-127), **23-bit mantissa**
- Double-precision (64 bits): 1-bit sign, 11-bit exponent with bias 1023 (excess-1023), and 52-bit mantissa
- Sign bit: 0 for + and 1 for -
- if qns states "no hidden bit", it means the leading 1 is also stored



MIPS - 32 bits instruction

R-format (op rd rs rt/ srl rd rt shamt)

1. opcode - 31-26
2. rs - 25-21
3. rt - 20-16
4. rd - 15-11
5. Shamt - 10-6
6. Funct - 6-0

Note srl/sll, rs = 0

I-format (op rs rt immediate)

1. opcode - 31-26
2. rs - 25-21
3. rt - 20-16
4. Immediate - 15-0

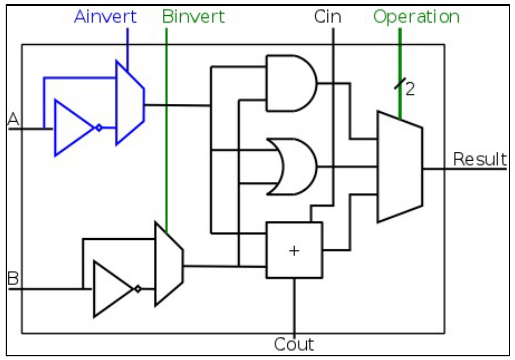
- bne & beq
  - Immediate value is the number of instructions from PC + 4 to the target branch
  - If target is before PC + 4, negative value
  - If target is after PC + 4, positive value

J-format (opcode address)

1. opcode - 31-26
2. Address - 26-0

- Memory addresses are 32 bits, to get 26,
  - First 4 bits of the 32 bit taken from PC + 4 address
  - Last 2 bits removed since addresses are word aligned i.e. divisible by 4
- Maximum jump range = 256MB = 2^{28}

Datapath & Control



- A/B invert inverts A/B to do subtraction using the + component
- Operation 2 bit is used to select 1 of the 3 results
- ALUop is the control signal; ALU control generates the 4 bit output using ALUop
- ALUcontrol3 = 0

ALUcontrol2 = ALUop0 + ALUop1.Funct[1]

Boolean Algebra

NOT > AND > OR

Identity laws

A+0 = 0+A = A

A.1 = 1.A = A

Inverse/Complement laws

A+A' = A'+A = 1

A.A'= A'.A = 0

Commutative laws

A+B = B+A

A.B = B.A

Associative laws

A+(B+C) = (A+B)+C = A+B+C

A.(B.C) = (A.B).C = A.B.C

Distributive laws

A.(B+C) = (A.B)+(A.C)

A+(B.C) = (A+B).(A+C)

Duality

Interchanging AND/OR operators → eqn still valid

E.g. distributive laws

Idempotency

X+X = X

X.X = X

One element / Zero element

X+1 = 1+X = 1

X.0 = 0.X = 0

Involution

(X')' = X

Absorption 1

X + X.Y = X

X . (X+Y) = X

Absorption 2

X + X'.Y = X + Y

X . (X'+Y) = X.Y

De Morgans' (can be generalised to >2 var)

(X+Y)' = X'.Y'

(X.Y)' = X'+Y'

Consensus

X.Y + X'.Z + Y.Z = X.Y + X'.Z

(X+Y).(X'+Z).(Y+Z) = (X+Y).(X'+Z)

Min terms and Max terms

- mx' = Mx
- F(X, Y, Z) = ∑m(1,4,5,6,7) = ∏M(0,2,3)
- F' = m0 + m2 + m3
- F = (m0 + m2 + m3)'= m0' · m2' · m3' = M0 · M2 · M3
- POS = SOP'

Useful equations

X'.Y + X.Y' = X⊕Y

X',Y' + X.Y = (X⊕Y)' = X⊙Y

Half-Adder

C = X.Y

S = X⊕Y

Full-Adder

C = X.Y + Z.(X⊕Y) = X.Y + X.Z + Y.Z

S = X ⊕ Y ⊕ Z = X'.Y'.Z + X'.Y.Z' + X.Y'.Z' + X.Y.Z

N-bit parallel adder

$C_{i+1}S_i = X_i + Y_i + C_i$

$C_{i+1} = X_i + Y_i + (X_i \oplus Y_i).C_i$

$S_i = X_i \oplus Y_i \oplus C_i$

Circuit Delay - n-bit ripple parallel adder

$S_n = [(n-1)*2 + 2]t$

$C_n = [(n-1)*2 + 3]t$

MSI Components

Decoder ($n \rightarrow 2^n$)

Output has only one 1 and other bits are 0

Encoder ($2^n \rightarrow n$)

- Input has only one 1 and other bits are 0
- Inputs with more than one 1 are invalid (X)
- Priority encoder
    - Allows input with more than one 1

Inputs				Outputs		
D_0	D_1	D_2	D_3	*f*	*g*	*V*
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Demultiplexer ($1 \rightarrow 2^n$)

- directs data from the input to one selected output line.
- n selectors

Multiplexer ($2^n \rightarrow 1$)

- Selects one input
- n selector

Sequential logic

J	K	Q(t+1)	Comments
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q(t)'	Toggle

S	R	Q(t+1)	Comments
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Unpredictable

D	Q(t+1)	
0	0	Reset
1	1	Set

T	Q(t+1)	
0	Q(t)	No change
1	Q(t)'	Toggle

Excitation Table

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Flip-flop

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

SR Flip-flop

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

D Flip-flop

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

T Flip-flop

M flip flops and n inputs = 2^{m+n} rows for table

Pipeline
Ideal clock cycles = (# instructions) + (n-stage - 1)
E.g. 5 stage pipeline → (# instructions) + 4

Structural Hazard
Only for instructions with data dependency!

Conditions	**Stage when data is ready**	**Number of cycles delayed**
Without Forwarding	MEM	+2
With Forwarding	EX	+0
With Forwarding (lw)	MEM	+1 (for inst after lw)

Control Hazard
Branch prediction: Why predict not taken is easier?
Predict not taken is easier to implement, as PC+4 is already computed. To implement predict taken, we need to compute the target branch address quickly in the first cycle. That requires additional hardware.

Conditions	**Stage when data is ready**	**Number of cycles delayed**
W/o forwarding, W/o early-branching w/ dependency e.g. add $t2 $t0 $t1 beq $t2 $s2 else	MEM	add → beq + 2 (delay at beq) beq → next + 3 (delay at next)
W/o forwarding, W/ early-branching w/ dependency	ID	add → beq + 2 beq → next + 1
W/ forwarding, W/ early-branching W/o data dependency	ID	add → beq + 0 beq → next + 1 Total: +1
W/ early-branching w/ data dependency	Data ready at EX but beq needs data at ID	add → beq +1 After beq +1 Total: +2
W/ early-branching (lw) E.g. lw $t2 $s2(8) beq $t2 $t0 else	Data ready at MEM but beq needs data at ID	lw → beq +2 After beq +1 Total: +3
Branch prediction without early branching		No delay unless incorrect prediction → flush 3 cycles; +3
Branch prediction with early branching		No delay unless incorrect prediction → flush 1 cycle; +1

Caching

1 byte = 8 bits
1 KB = 2^{10} bytes
1 MB = 2^{20} bytes
1 GB = 2^{30} bytes

Average Access Time = Hit rate x Hit Time + (1-Hit rate) x Miss penalty
E.g. Suppose our on-chip SRAM (cache) has 0.8 ns access time, but the fastest DRAM (main memory) we can get has an access time of 10ns. How high a hit rate do we need to sustain an average access time of 1ns?
Let h be the desired hit rate.
$1 = 0.8h + (1 - h) \times (10 + 0.8) = 0.8h + 10.8 - 10.8h$
$10h = 9.8 \rightarrow h = 0.98$ Hence we need a hit rate of **98%**.

Offset = $\log_2$(no. of words in a block * word size) = $\log_2$(bytes in each block)
Cache index = $\log_2$(number of blocks in cache) = $\log_2$(cache capacity/block capacity)

1. Direct-mapped
    - Has 1 exact spot in the cache
2. N-set associative
    - N spots in the cache
3. Fully associative cache
    - Any spot in the cache

Least recently used (LRU)

Write policy

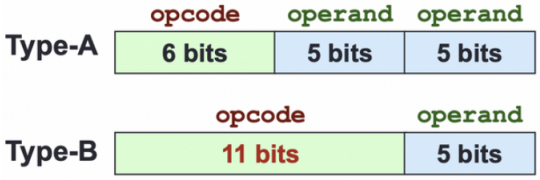
1. Write through - write data to cache and memory
    a. Problem: write will operate at the speed of main memory
    b. Solution: write buffer
2. Write-back - only write to cache, write to memory when cache is replaced
    a. Problem: wasteful if we write back every replaced block
    b. Solution: add dirty bit
        i. Change dirty bit to 1 if writing
        ii. Only write back if dirty bit is 1

Write miss
If the block being written is not in cache

1. Write allocate
    a. Load block into cache and write to cache
    b. Write back depends on write policy
    c. Write around
        i. Write directly to memory

Max & Min instructions

What is the maximum number of instructions?



Answer:

$1 + (2^6 - 1) \times 2^5$

$= 1 + 63 \times 32$

$= \mathbf{2017}$

Reasoning:

1. For every 6-bit prefix (front-part) given to Type-B, we get 2^5 unique patterns, e.g. `[111111]xxxxx`
2. So, we should minimize Type-A instruction and give as many 6-bit prefixes as possible to Type-B

   ➔ 1 Type-A instruction, $2^6 - 1$ prefixes for Type-B

- Design an expanding opcode for the following to be encoded in a 36-bit instruction format. An address takes up 15 bits and a register number 3 bits.
  - 7 instructions with two addresses and one register number.
  - 500 instructions with one address and one register number.
  - 50 instructions with no address or register.

One possible answer:

3 bits	15 bits	15 bits	3 bits
000 → 110 opcode	address	address	register
111	000000 + 9 bits opcode	address	register
111	000001 ⋮ 110010 + 9 0s opcode	unused	unused

An ISA has 16-bit instructions and 5-bit addresses. There are two classes of instructions: class *A* instructions have one address, while class *B* instructions have two addresses. Both classes exist and the encoding space for opcode is completely utilised.

▼ (a) What is the minimum total number of instructions?

Class *A*: `ppppppppppp xxxxx`
Class *B*: `qqqqqq xxxxx yyyyy`
`ppppppppppp` and `qqqqqq` are opcodes, `xxxxx` and `yyyyy` are addresses.

- class *B* $(2^6 - 1)$ opcodes, leaving one 6-bit opcode as the prefix for class *A* 11-bit opcodes.
- class *A* has (2^5) opcodes.
- $Total = (2^6 - 1) + (2^5) = 95$

▼ (b) What is the maximum total number of instructions?

- give class B only 1 opcode, leaving $(2^6 - 1)$ prefixes for class A opcodes.
- class A has $(2^6 - 1)(2^5) = 2016$ opcodes.
- $Total = 2016 + 1 = 2017$

Opcode	ALUop	Instruction Operation	Funct field	ALU action	ALU control
lw	00	load word	xxxxxx	add	0010
sw	00	store word	xxxxxx	add	0010
beq	01	branch equal	xxxxxx	subtract	0110
R-type	10	add	10 0000	add	0010
R-type	10	subtract	10 0010	subtract	0110
R-type	10	AND	10 0100	AND	0000
R-type	10	OR	10 0101	OR	0001
R-type	10	set on less than	10 1010	set on less than	0111

Generation of 2-bit **ALUop** signal will be discussed later

Instruction Type	ALUop
lw / sw	00
beq	01
R-type	10

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Big-endian:

Most significant byte stored in lowest address.

Example:

IBM 360/370, Motorola 68000, MIPS (Silicon Graphics), SPARC.

Example: **0xDE AD BE EF**

Stored as:

0	DE
1	AD
2	BE
3	EF

Little-endian:

Least significant byte stored in lowest address.

Example:

Intel 80x86, DEC VAX, DEC Alpha.

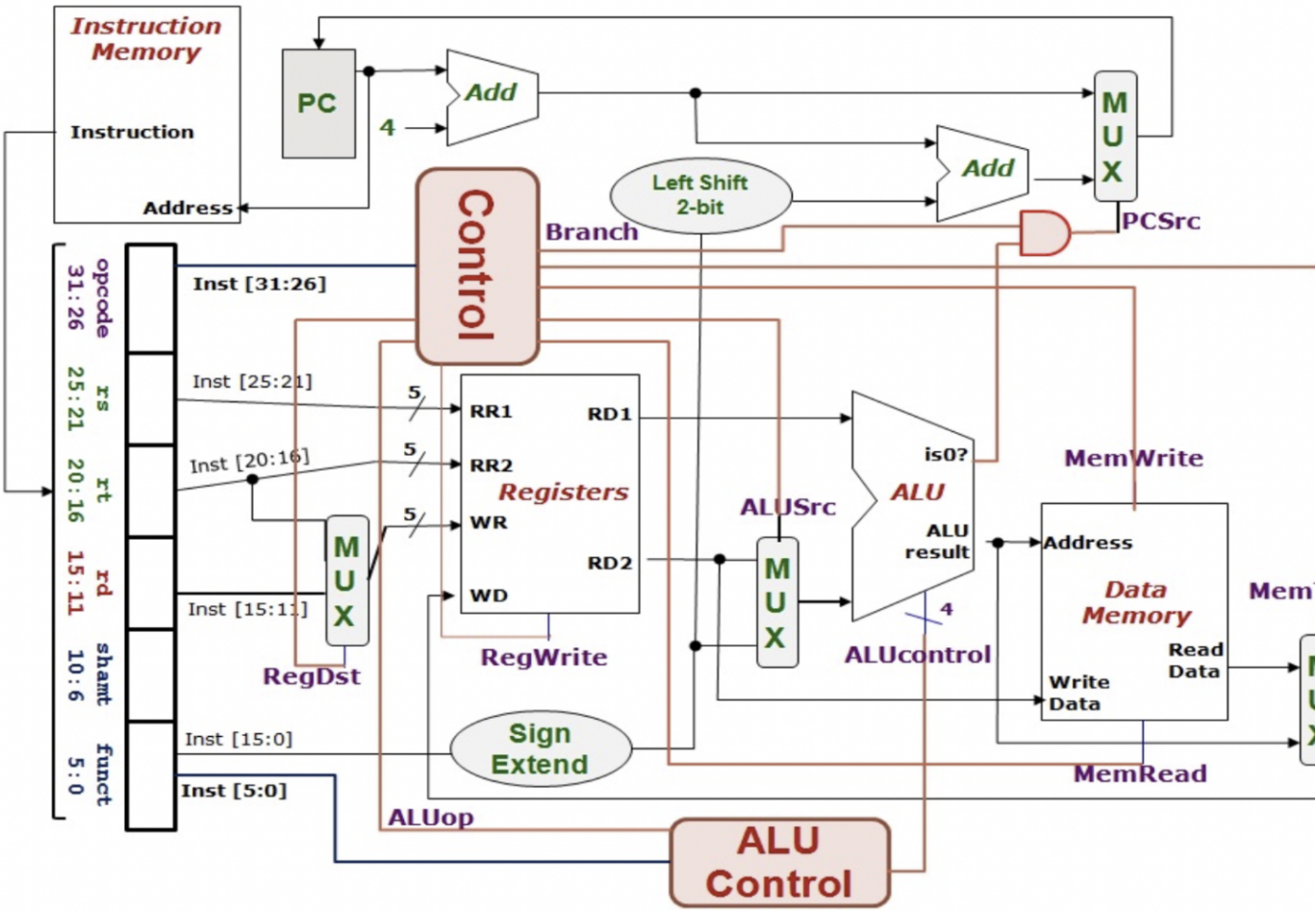
Example: **0xDE AD BE EF**

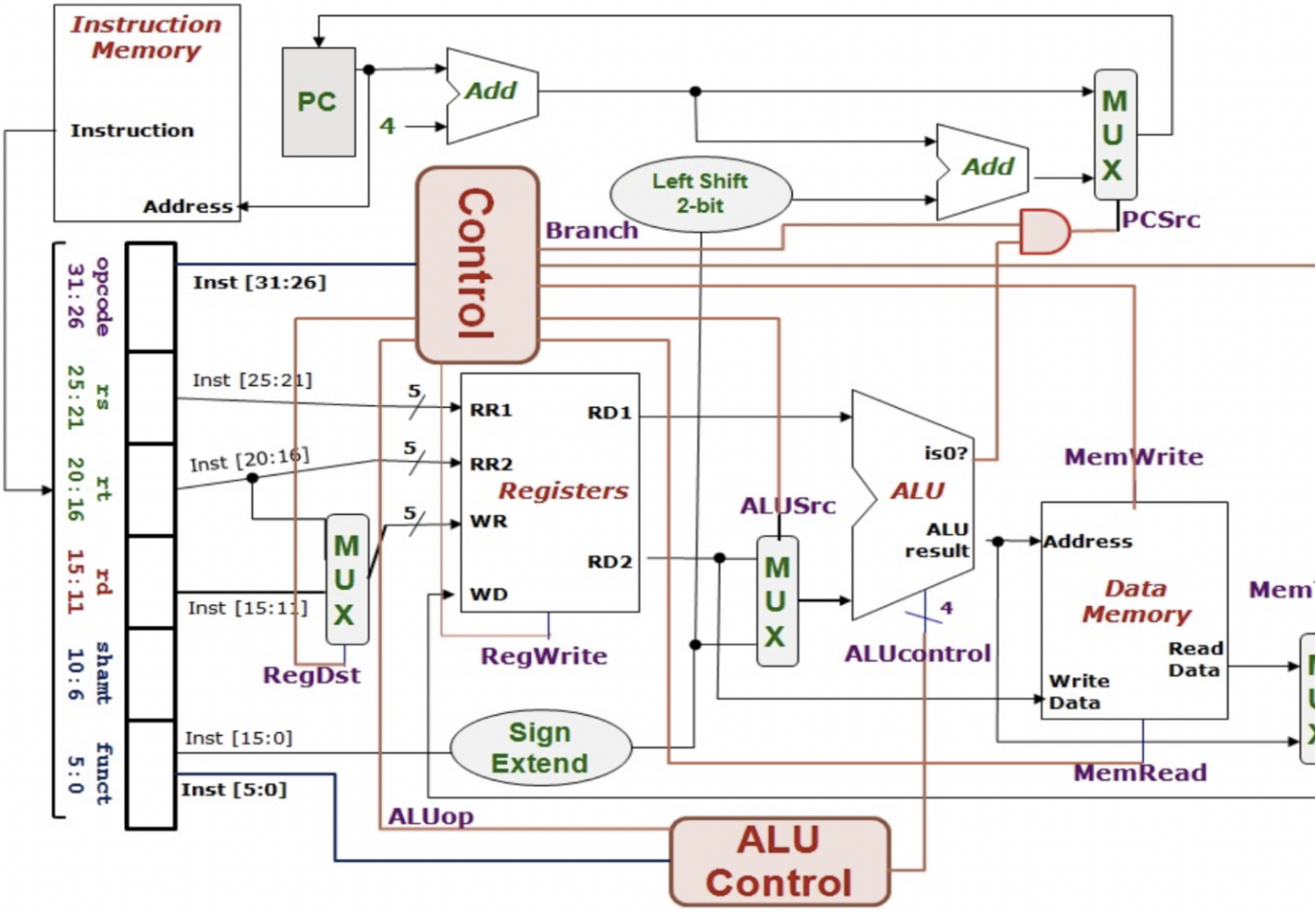
Stored as:

0	EF
1	BE
2	AD
3	DE

	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Stack	Accumulator	Register (load-store)	Memory-Memory
Push A	Load A	Load R1,A	Add C, A, B
Push B	Add B	Load R2,B	
Add	Store C	Add R3,R1,R2	
Pop C		Store R3,C	


Instruction Memory
Instruction
Address
PC
4
Add
Add
MUX
Left Shift 2-bit
Branch
PCSrc
Control
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
Inst [31:26]
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
Inst [5:0]
5
RR1
RR2
WR
WD
RD1
RD2
Registers
RegDst
RegWrite
ALUSrc
is0?
ALU
ALU result
4
ALUcontrol
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
Sign Extend
ALUop
ALU Control

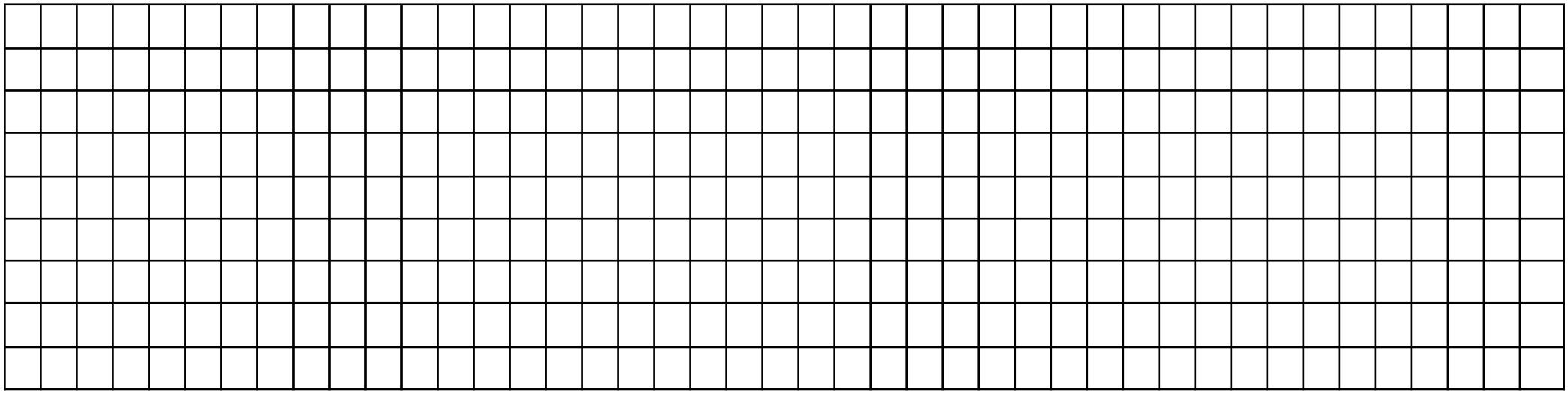

Instruction Memory
Instruction
Address
PC
4
Add
Add
MUX
Left Shift 2-bit
Branch
PCSrc
Control
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
Inst [31:26]
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
Inst [5:0]
5
RR1
RR2
WR
WD
RD1
RD2
Registers
RegDst
RegWrite
ALUSrc
is0?
ALU
ALU result
4
ALUcontrol
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
Sign Extend
ALUop
ALU Control

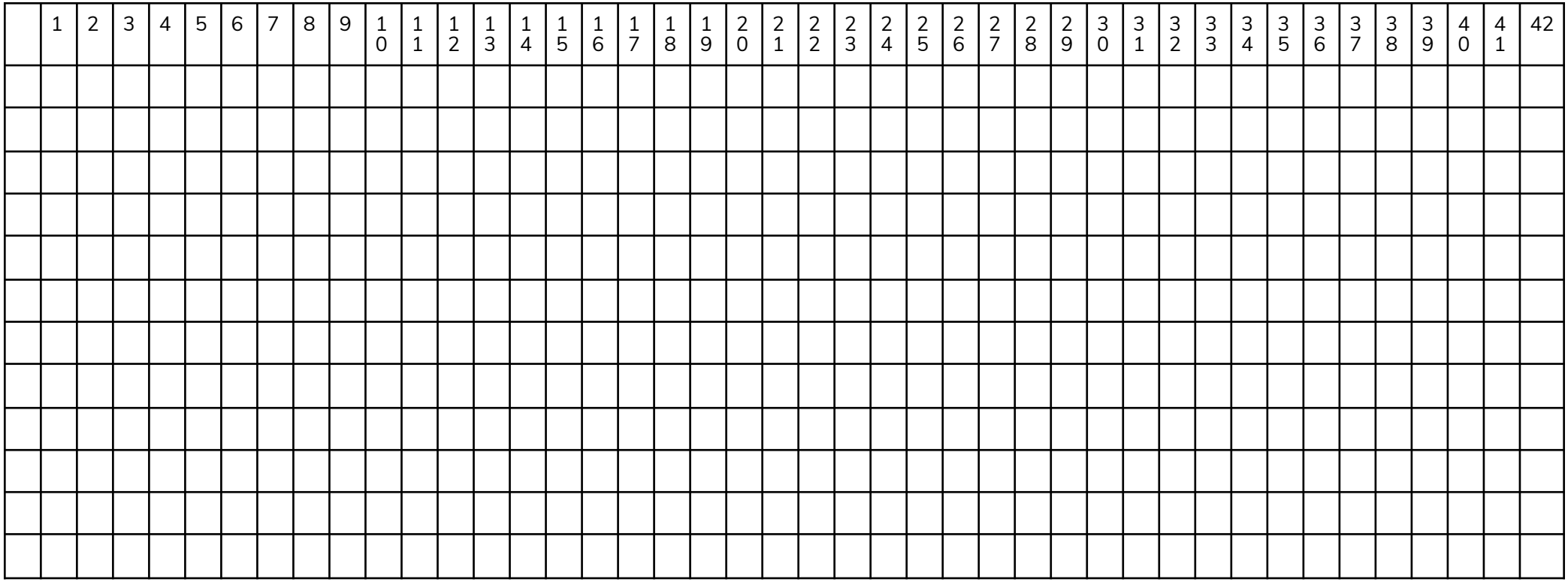
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
1																																									
2																																										
3																																									
4																																									
5																																									
6																																									
7																																									
8																																									
9																																									
10																																									
11																																									
12																																									
13																																									
14																																									
15																																									
16																																									
17																																									
18																																									
19																																									
20																																									
21																																									

22																																										
23																																										
24																																										

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42




1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42