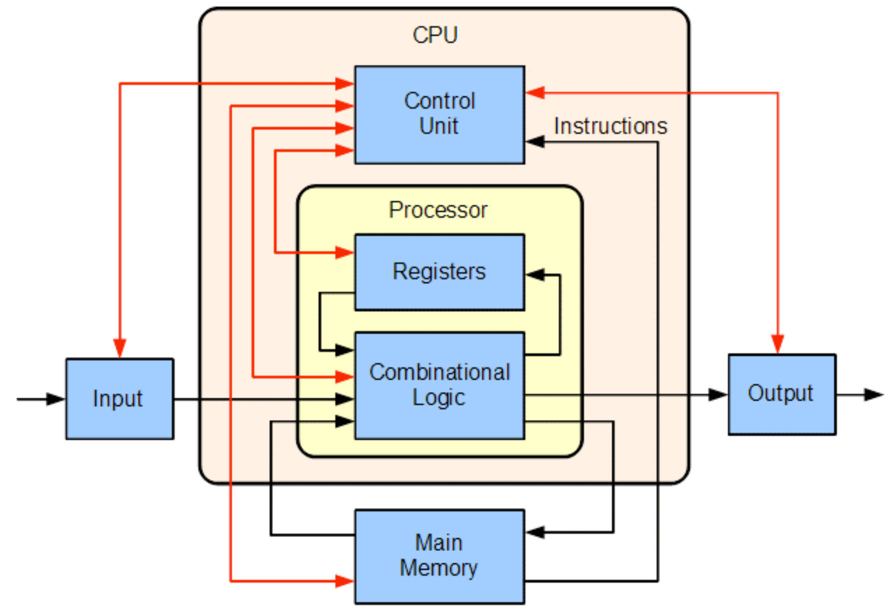
CSC 411

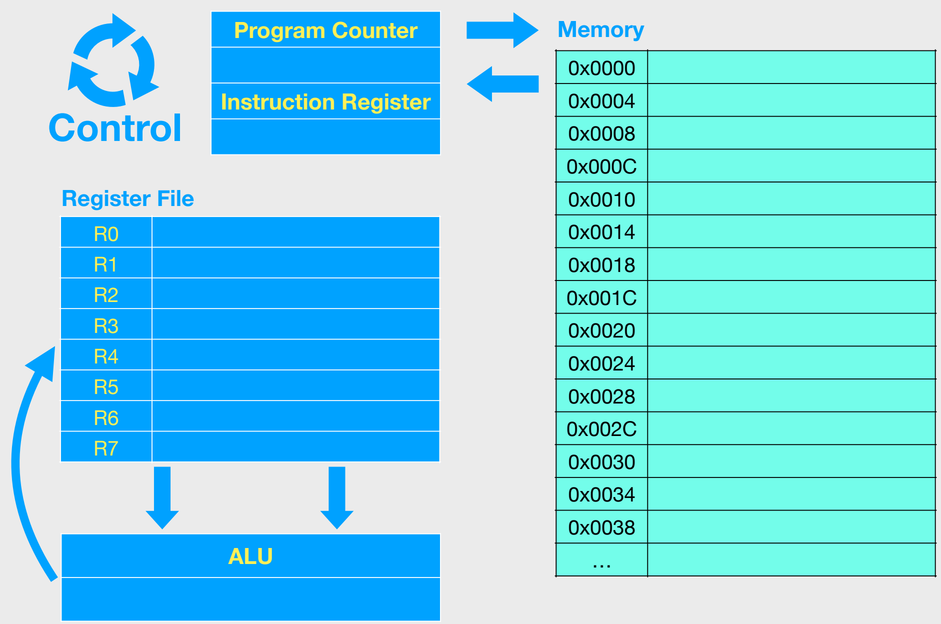
Computer Organization (Spring 2022)
Lecture 6: Executing Instructions and Integer Representation

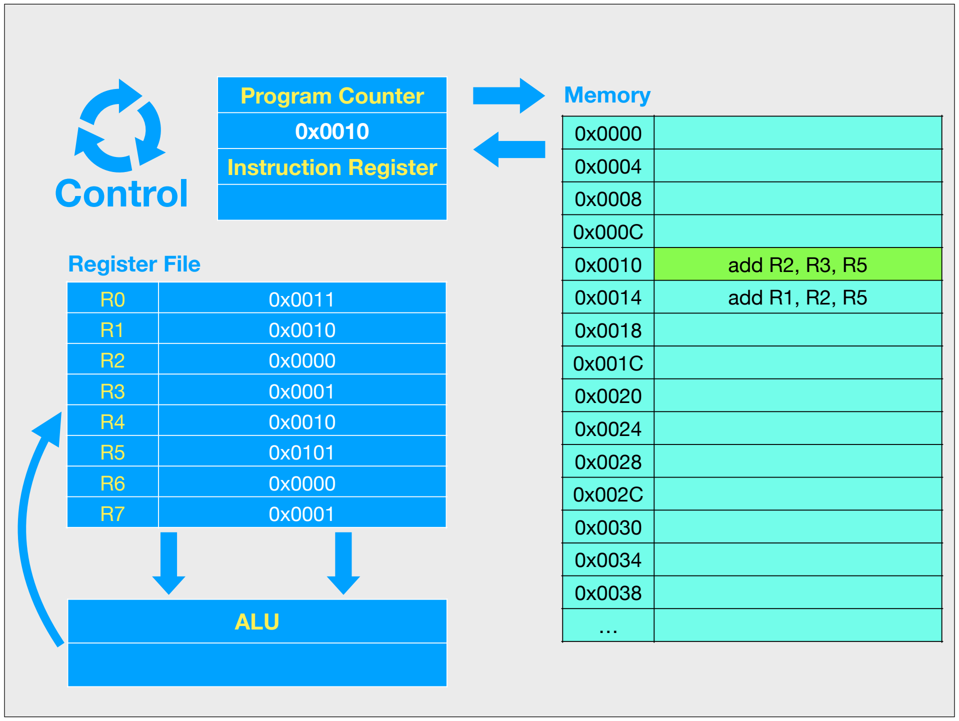
Prof. Marco Alvarez, University of Rhode Island

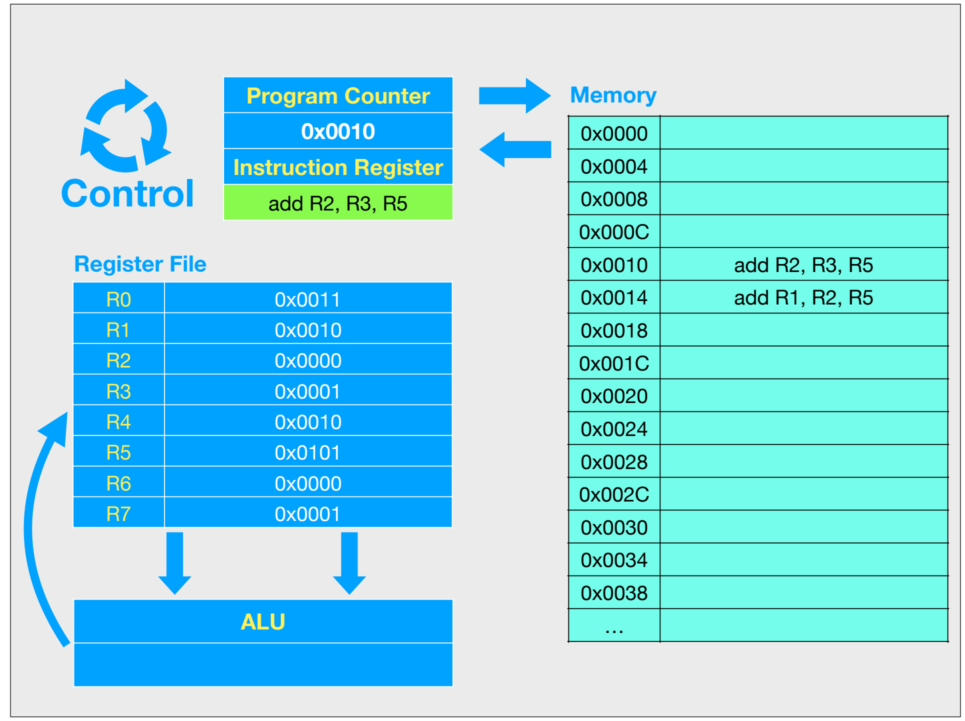


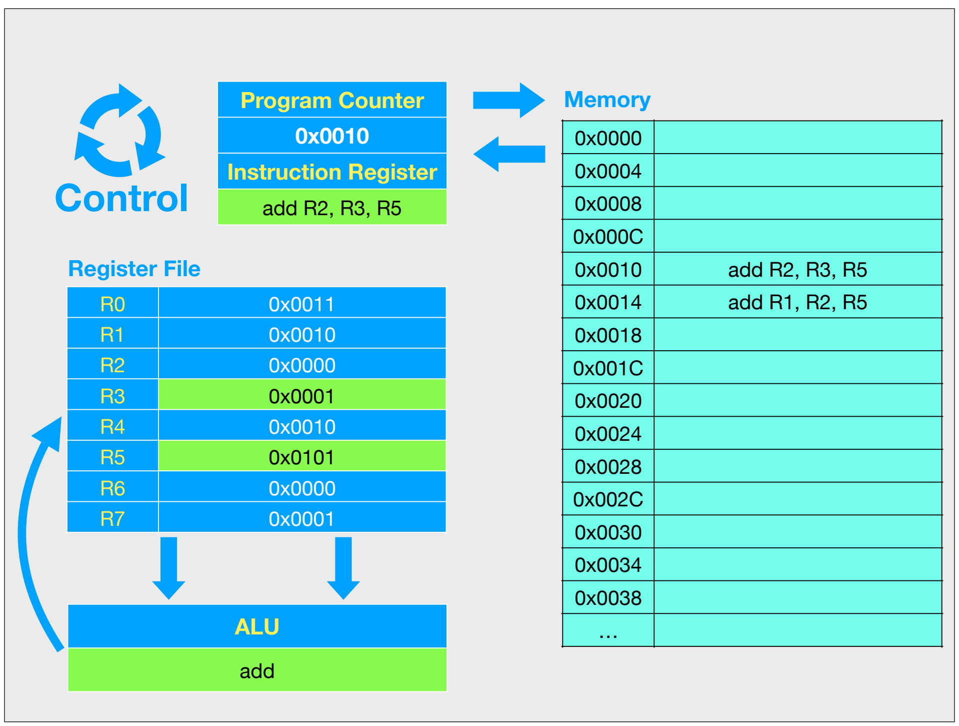
https://en.wikipedia.org/wiki/Computer_architecture

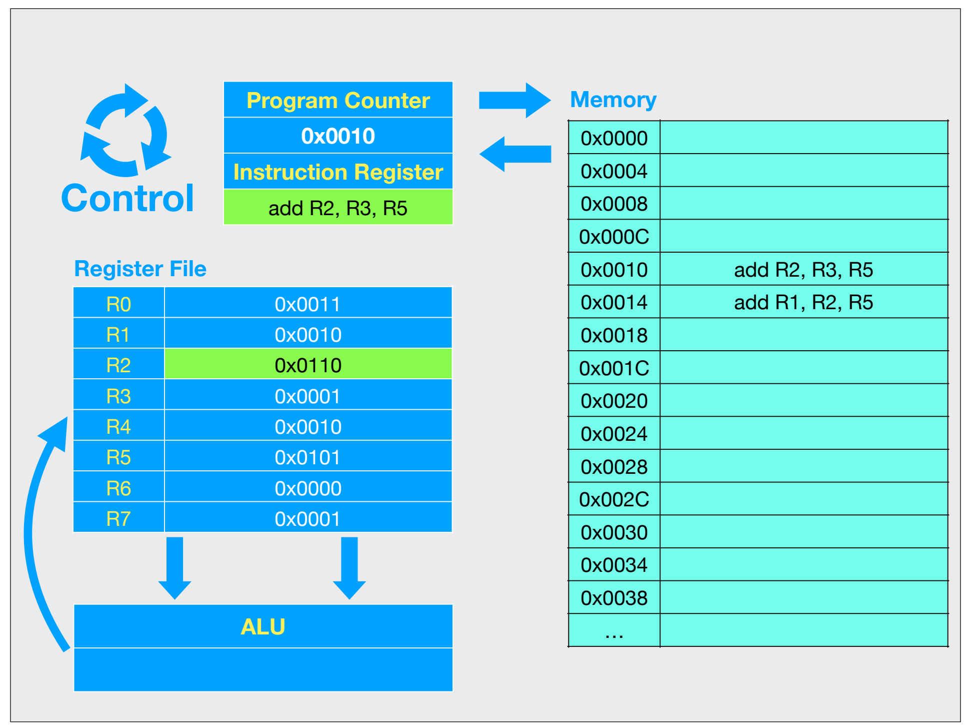
Executing instructions

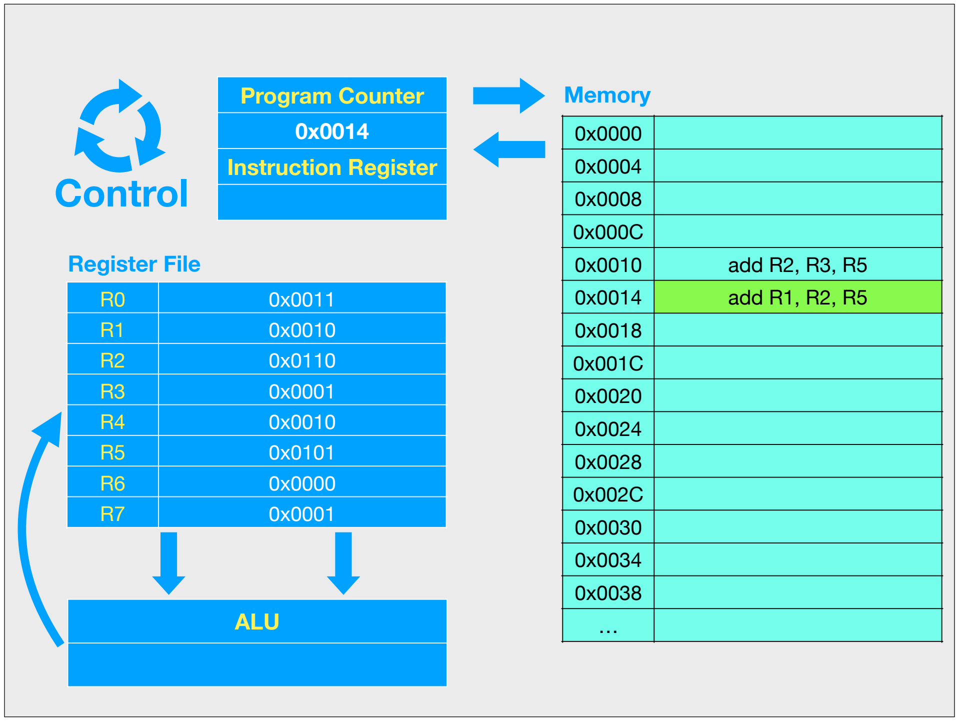


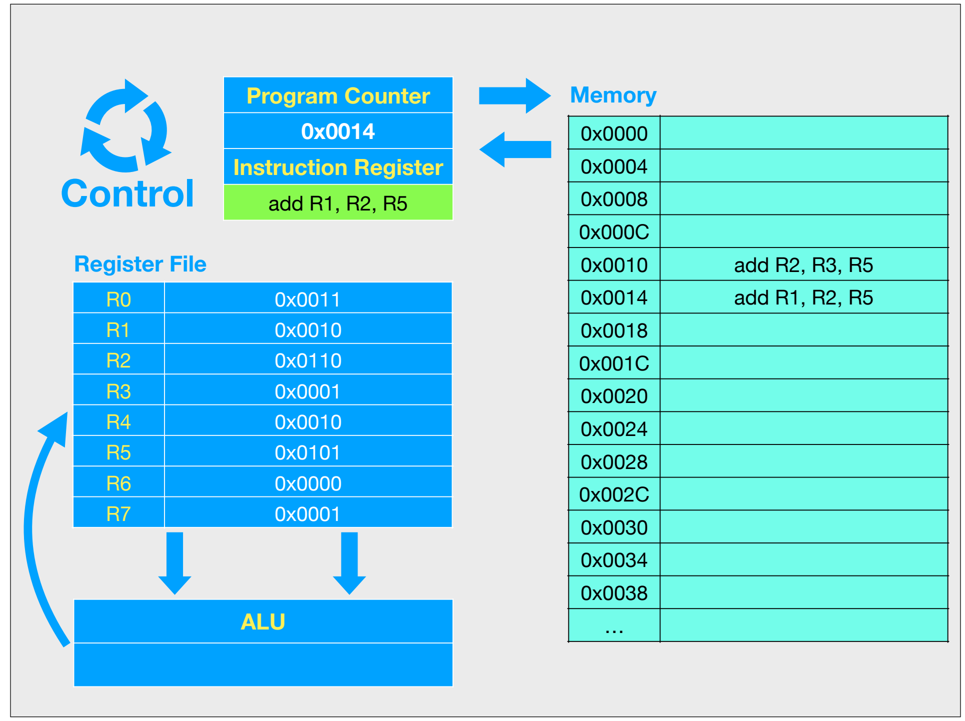

Control
Program Counter
0x0010
Instruction Register
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...
Register File
R0 0x0011
R1 0x0010
R2 0x0000
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU

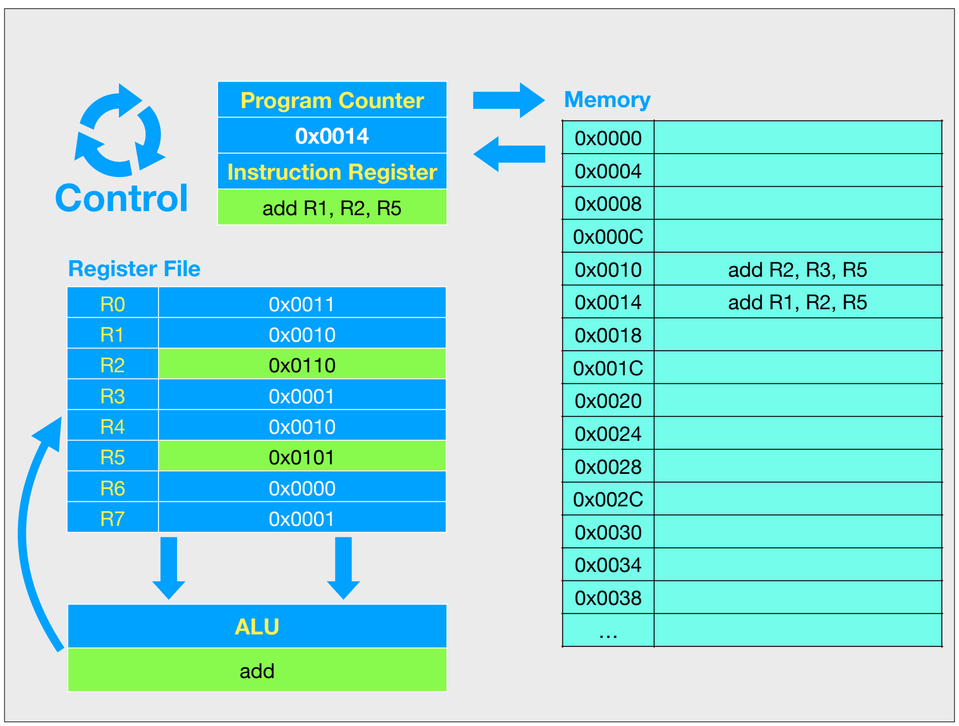

Control
Program Counter
0x0010
Instruction Register
add R2, R3, R5
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...
Register File
R0 0x0011
R1 0x0010
R2 0x0000
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU

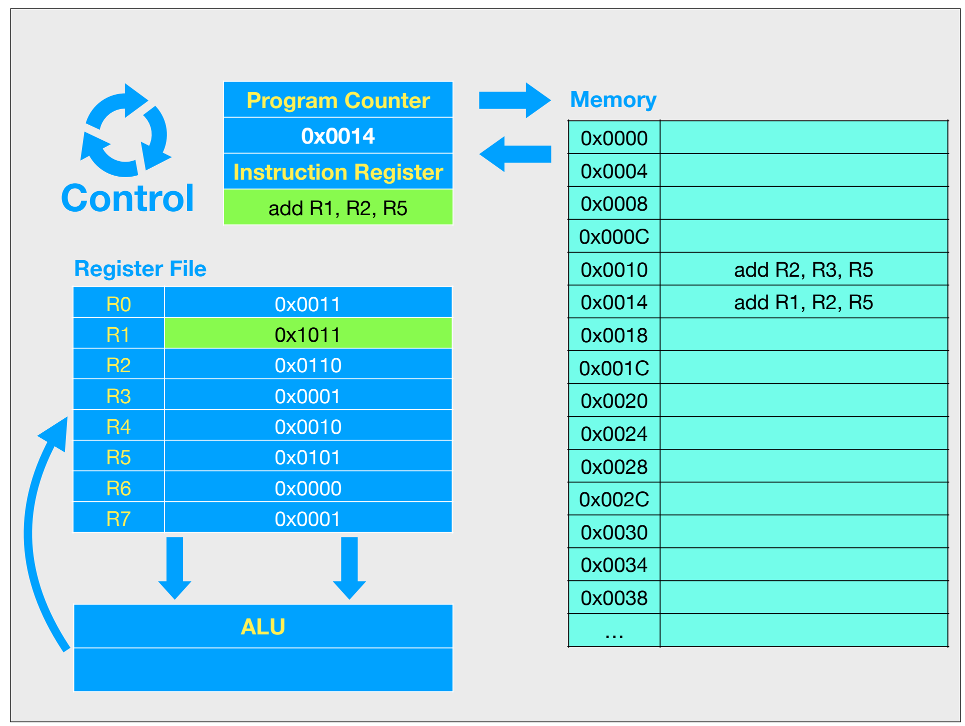

Control
Program Counter
0x0010
Instruction Register
add R2, R3, R5
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...
Register File
R0 0x0011
R1 0x0010
R2 0x0000
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU
add


Control
Program Counter
0x0010
Instruction Register
add R2, R3, R5
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...
Register File
R0 0x0011
R1 0x0010
R2 0x0110
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU

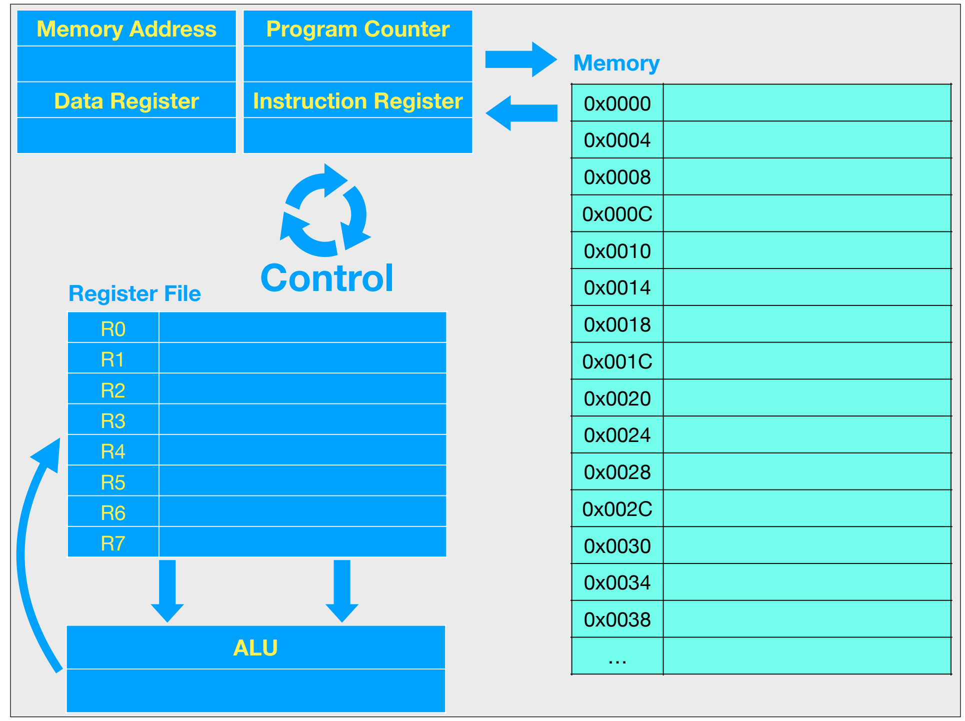

Control
Program Counter
0x0014
Instruction Register
Register File
R0 0x0011
R1 0x0010
R2 0x0110
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...

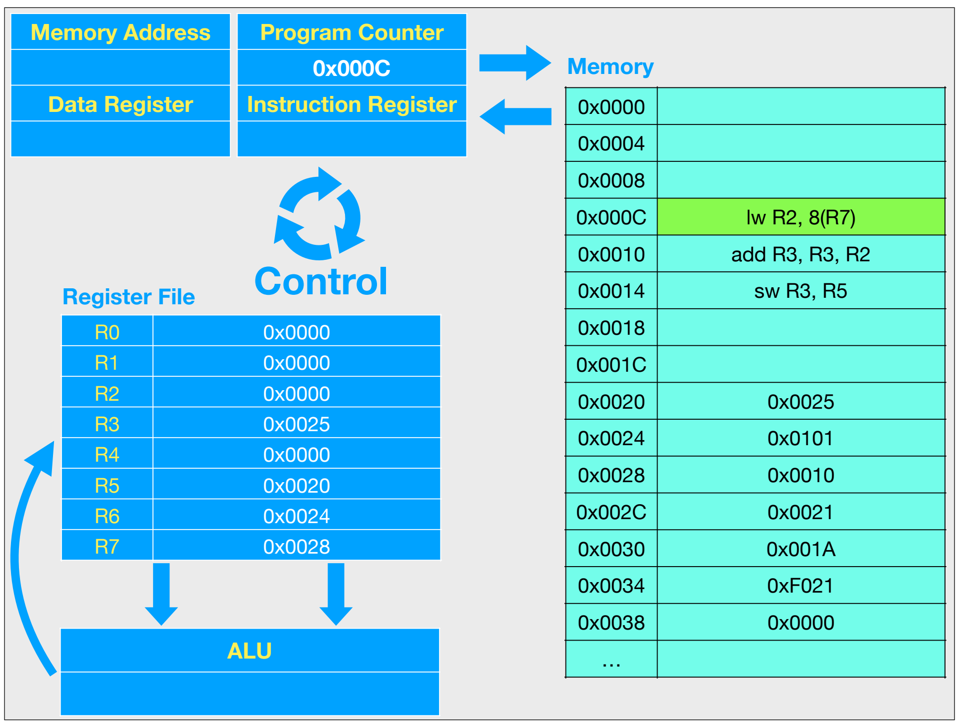

Control
Program Counter
0x0014
Instruction Register
add R1, R2, R5
Register File
R0 0x0011
R1 0x0010
R2 0x0110
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...

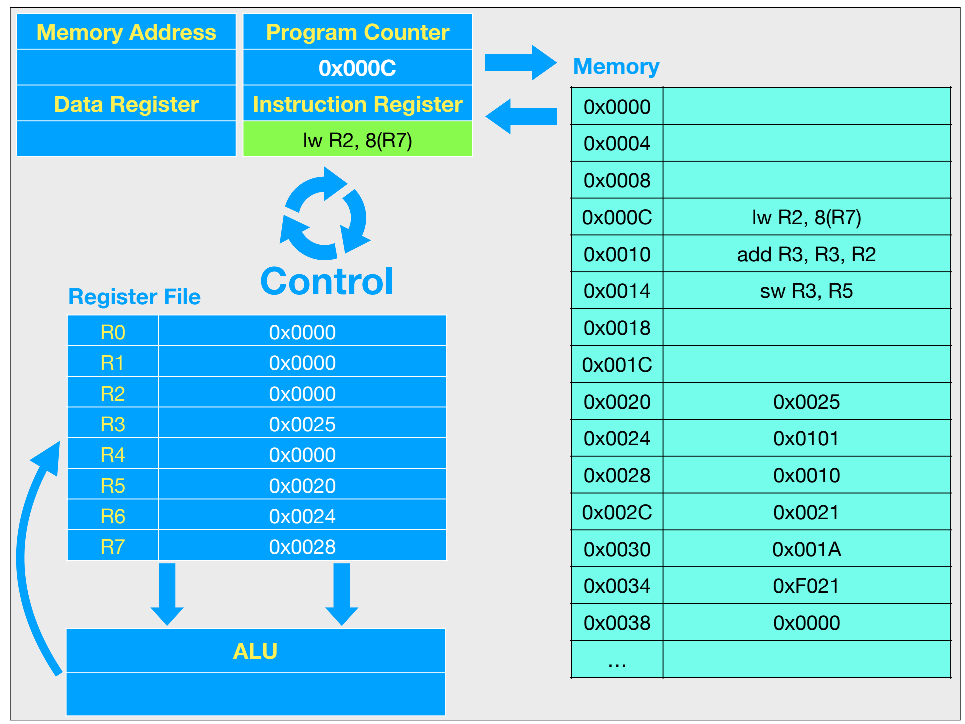

Control
Program Counter
0x0014
Instruction Register
add R1, R2, R5
Register File
R0 0x0011
R1 0x0010
R2 0x0110
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU
add
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...

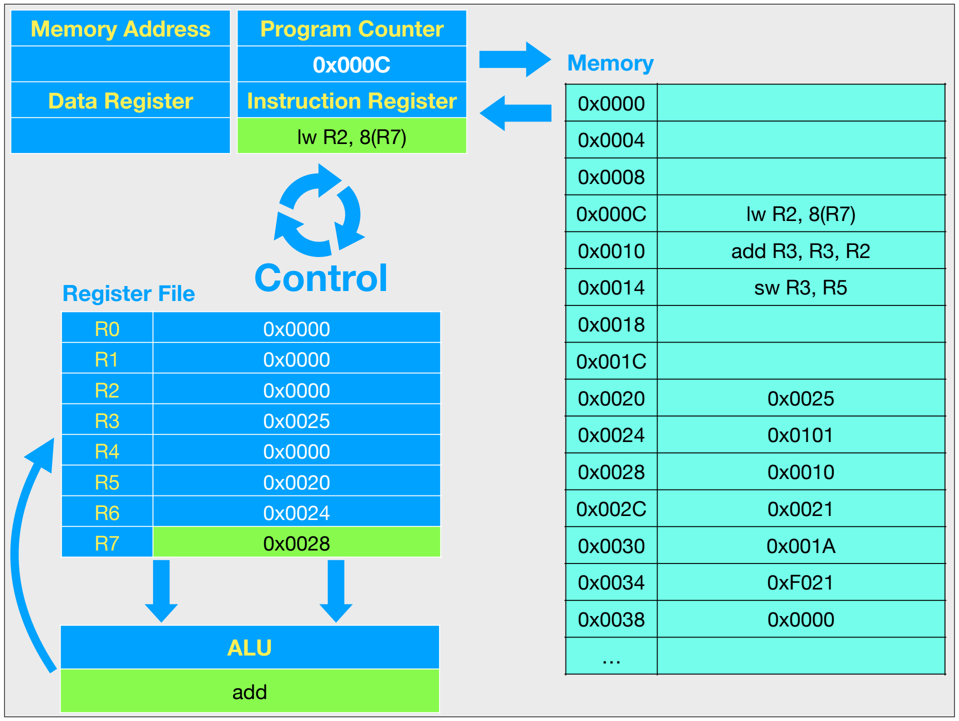

Control
Program Counter
0x0014
Instruction Register
add R1, R2, R5
Register File
R0 0x0011
R1 0x1011
R2 0x0110
R3 0x0001
R4 0x0010
R5 0x0101
R6 0x0000
R7 0x0001
ALU
Memory
0x0000
0x0004
0x0008
0x000C
0x0010 add R2, R3, R5
0x0014 add R1, R2, R5
0x0018
0x001C
0x0020
0x0024
0x0028
0x002C
0x0030
0x0034
0x0038
...

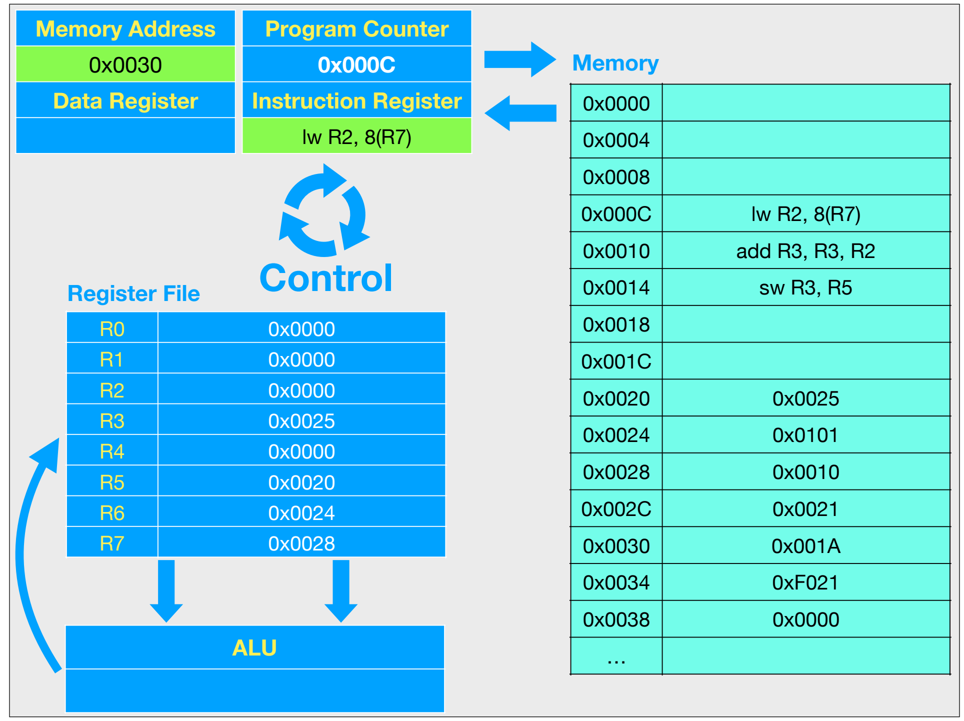
Executing memory instructions

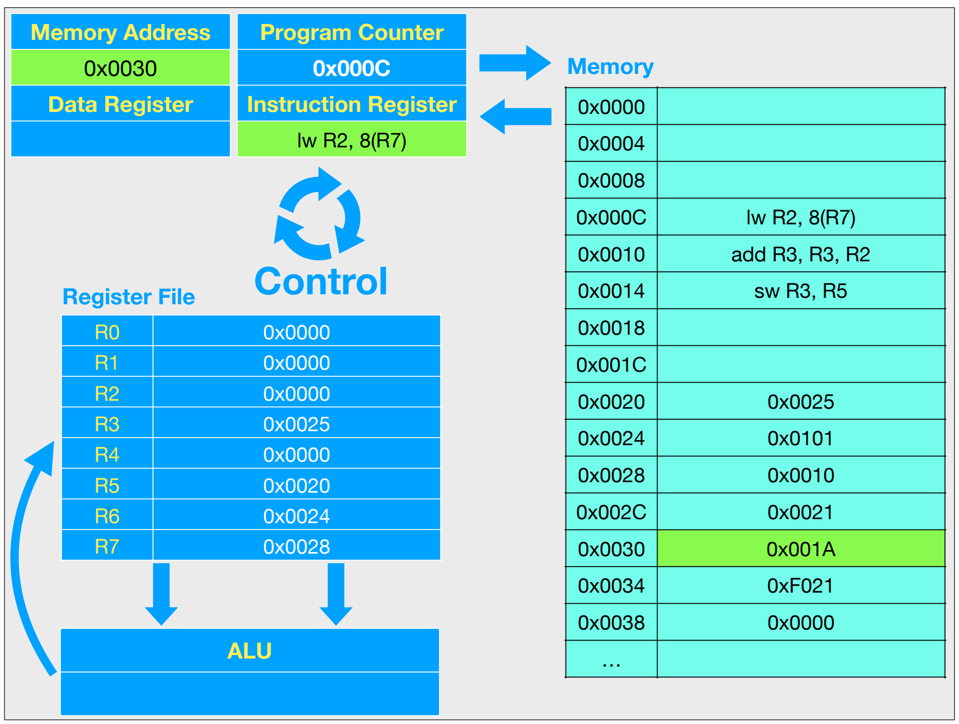


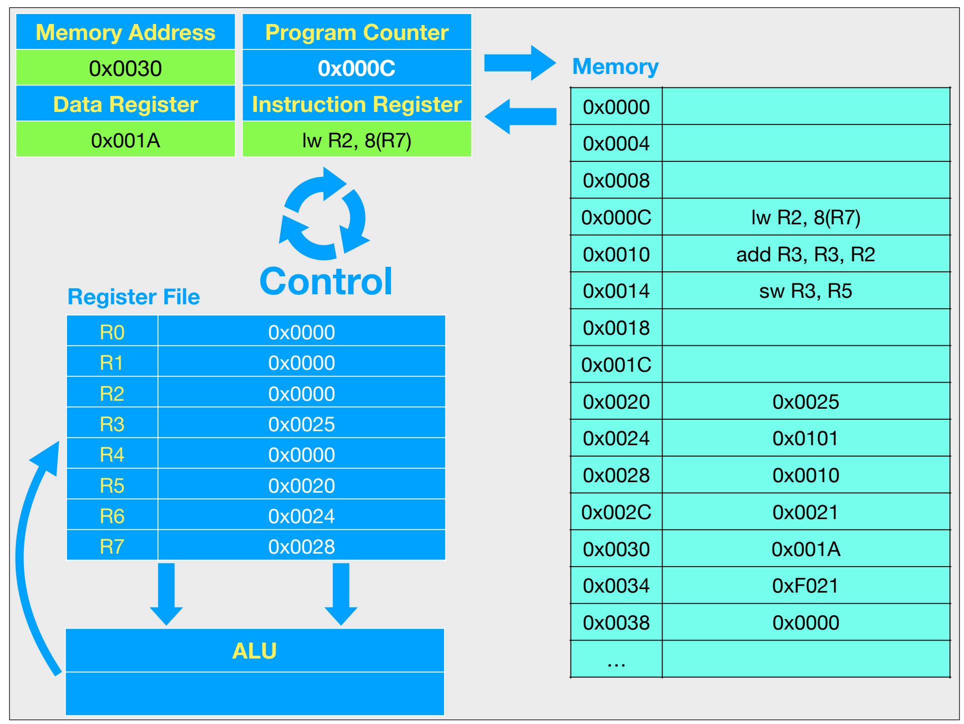


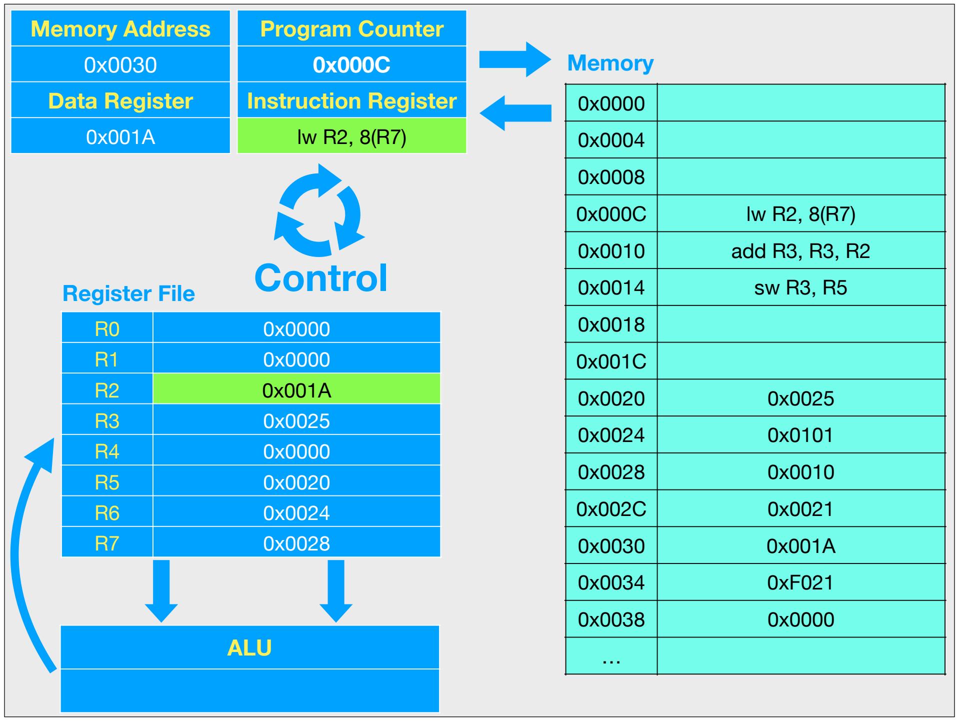


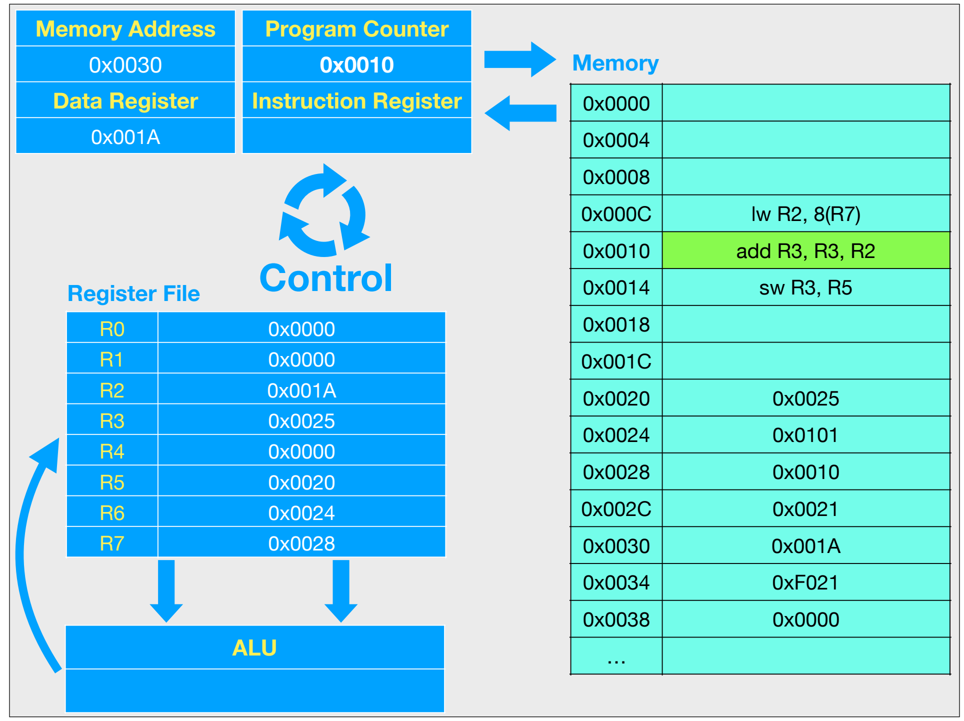

Memory Address
Program Counter
0x000C
Data Register
Instruction Register
lw R2, 8(R7)
Control
Register File
R0 0x0000
R1 0x0000
R2 0x0000
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
add
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
…

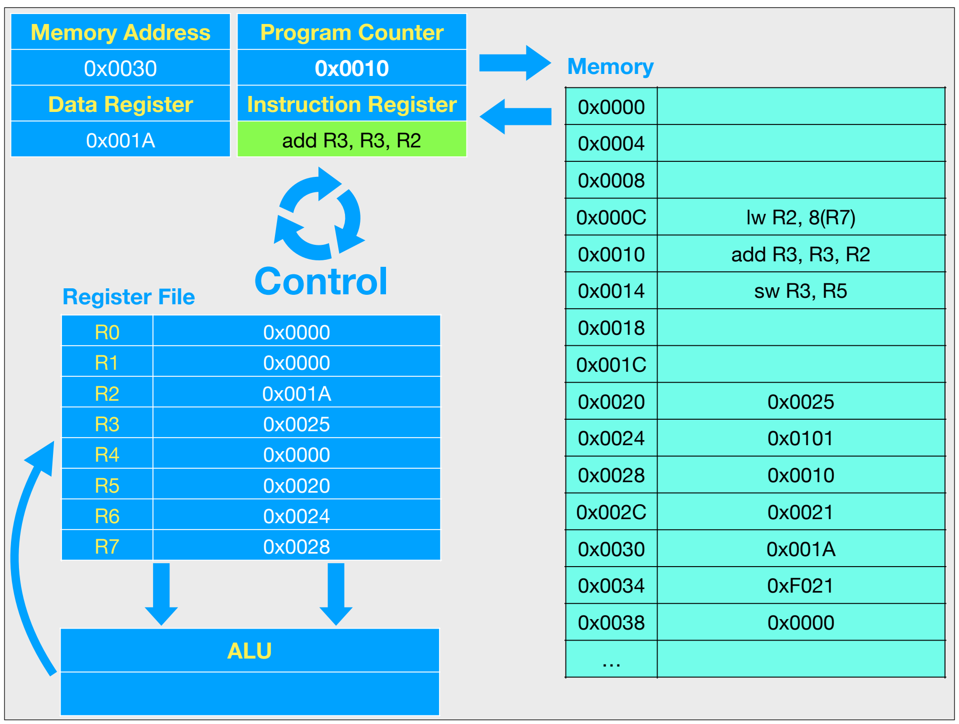

Memory Address
0x0030
Program Counter
0x000C
Data Register
Instruction Register
lw R2, 8(R7)
Control
Register File
R0 0x0000
R1 0x0000
R2 0x0000
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
…

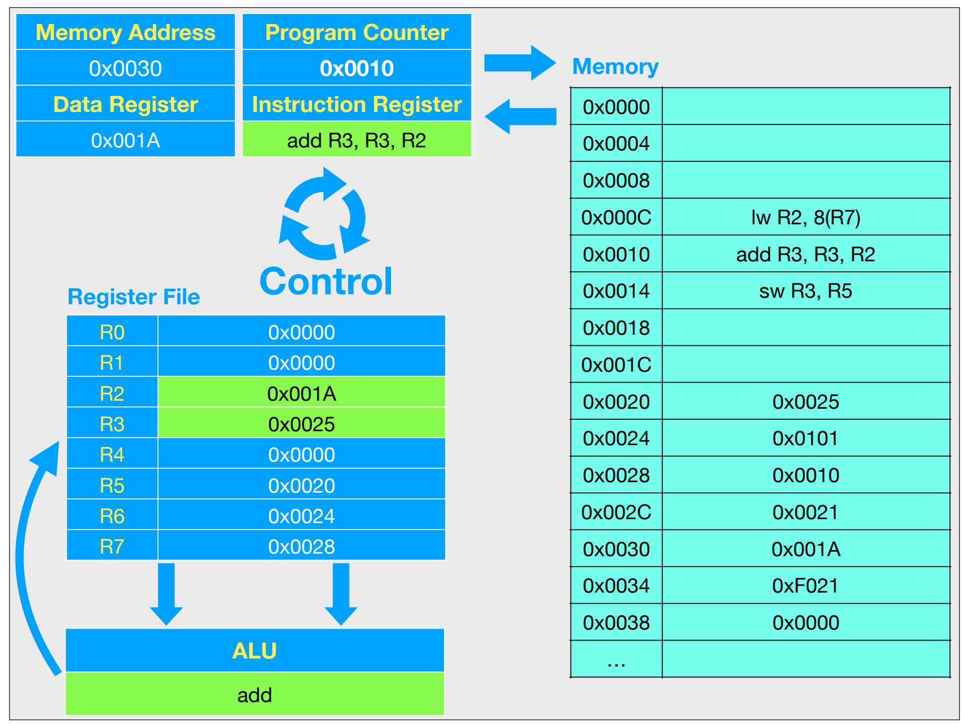

Memory Address
0x0030
Program Counter
0x000C
Data Register
Instruction Register
lw R2, 8(R7)
Control
Register File
R0 0x0000
R1 0x0000
R2 0x0000
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
…

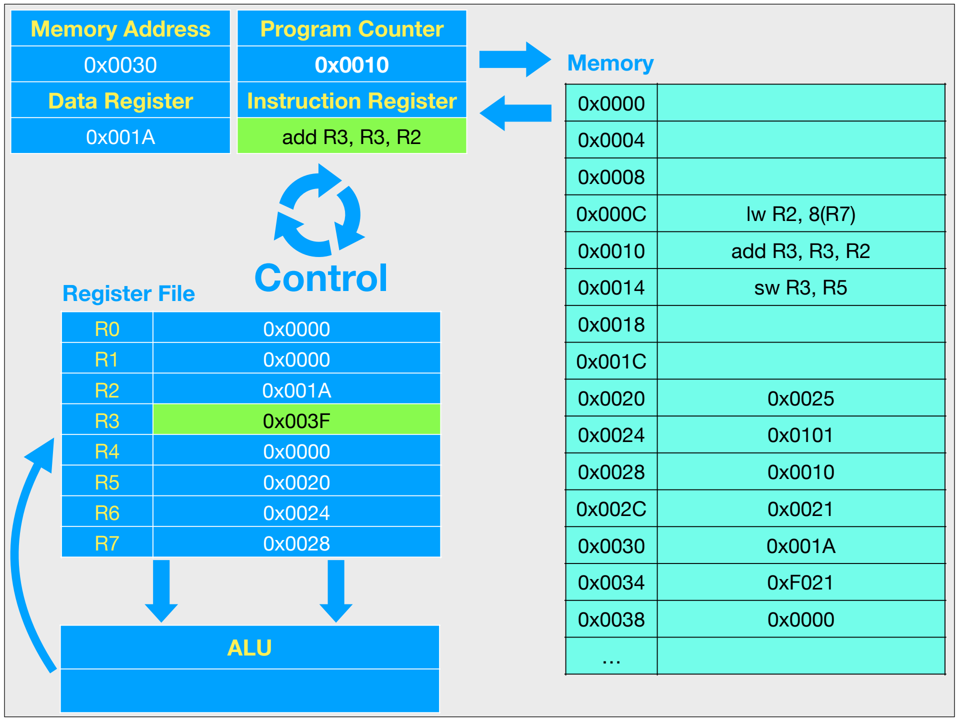

Memory Address
0x0030
Program Counter
0x000C
Data Register
0x001A
Instruction Register
lw R2, 8(R7)
Control
Register File
R0 0x0000
R1 0x0000
R2 0x0000
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
…

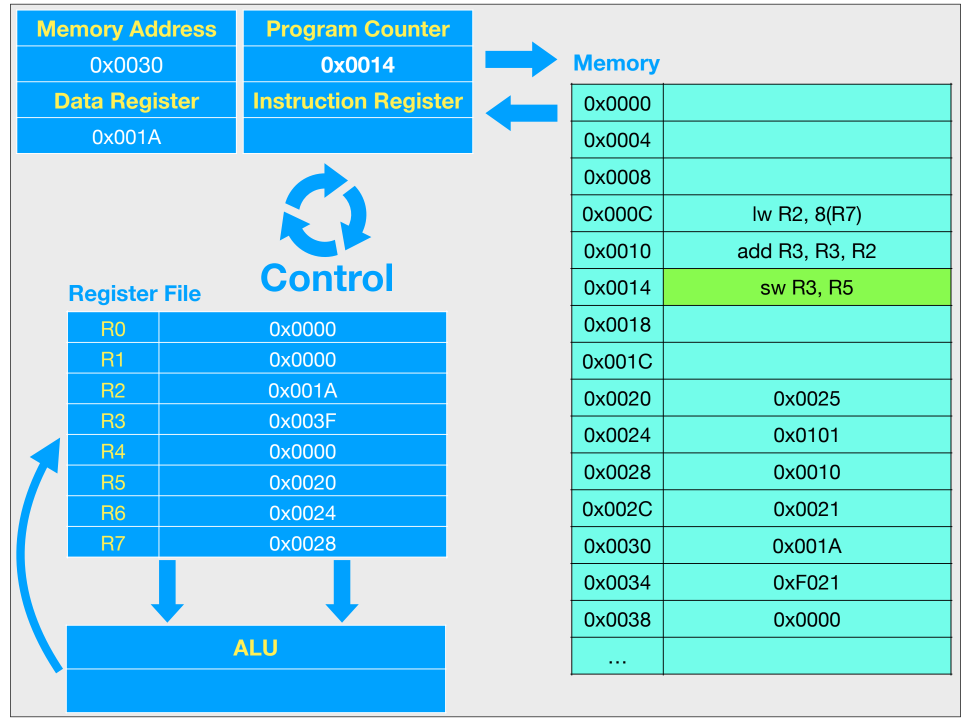

Memory Address
0x0030
Program Counter
0x000C
Data Register
0x001A
Instruction Register
lw R2, 8(R7)
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU

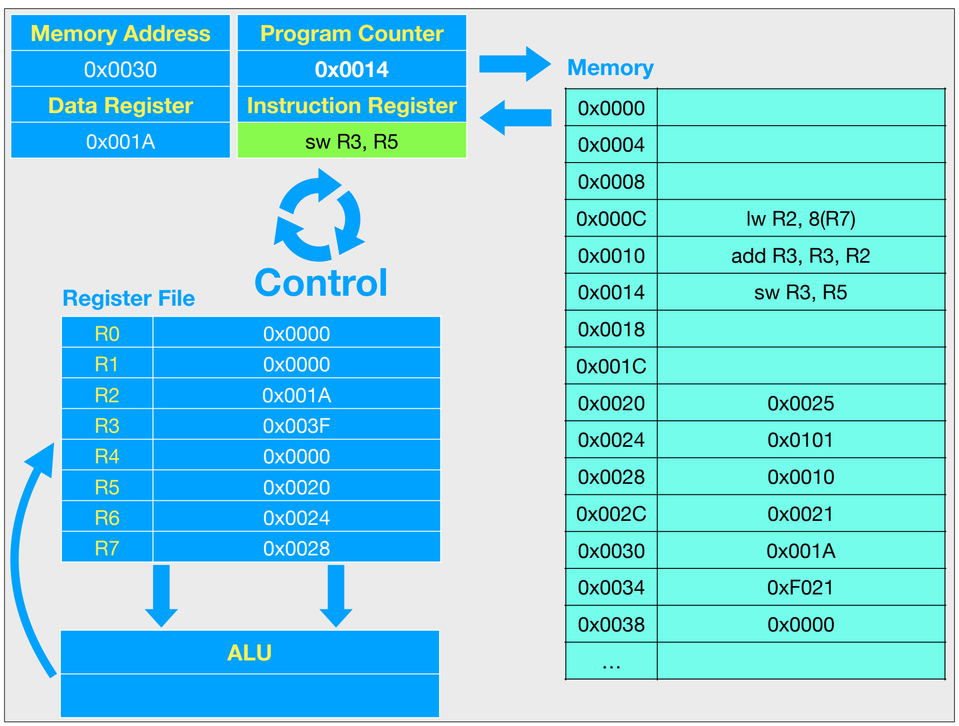

Memory Address
0x0030
Program Counter
0x0010
Data Register
0x001A
Instruction Register
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU

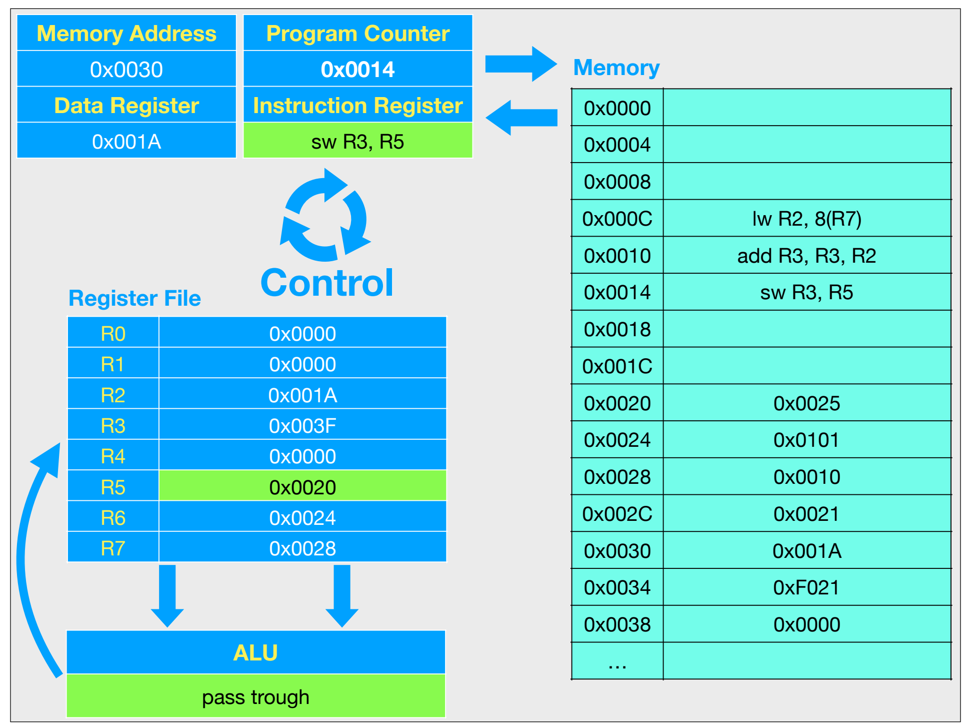

Memory Address
0x0030
Program Counter
0x0010
Data Register
0x001A
Instruction Register
add R3, R3, R2
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU

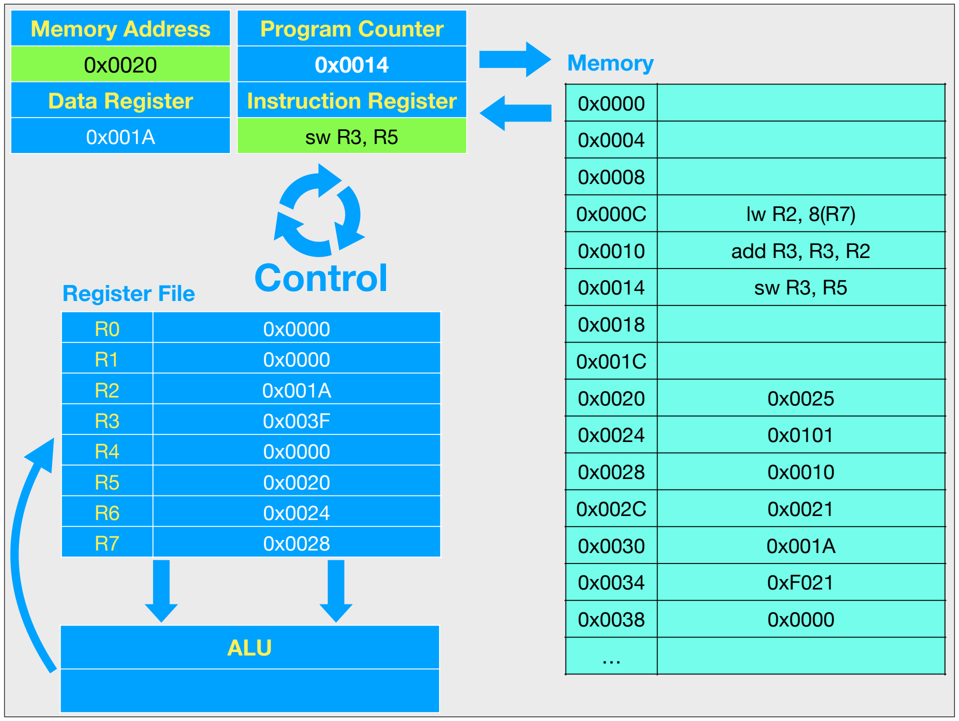

Memory Address
0x0030
Program Counter
0x0010
Data Register
0x001A
Instruction Register
add R3, R3, R2
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x0025
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
add

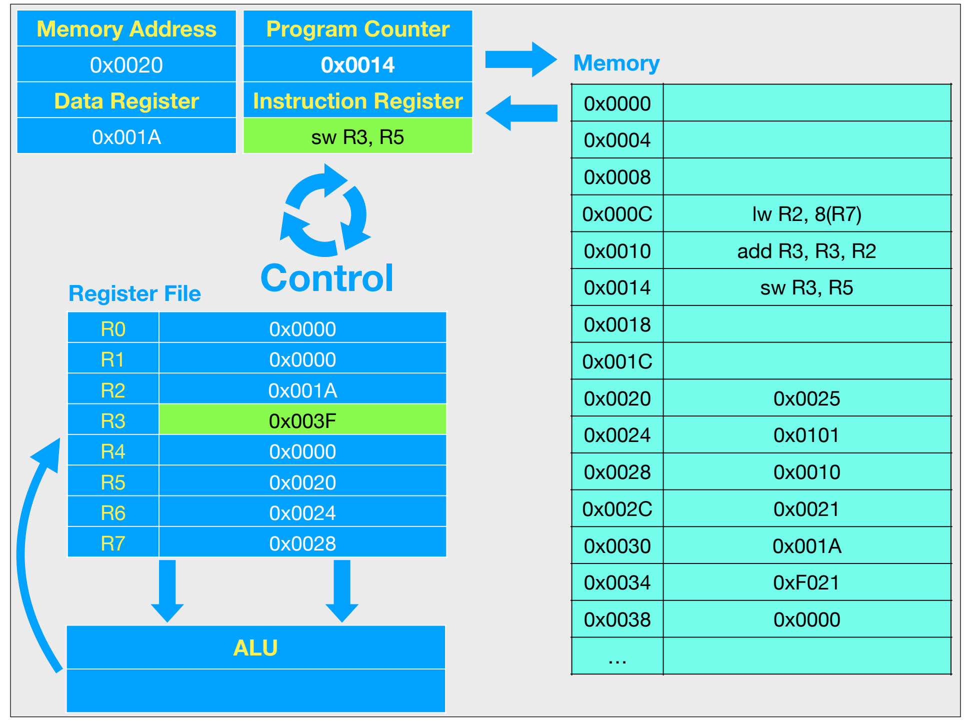

Memory Address
0x0030
Program Counter
0x0010
Data Register
0x001A
Instruction Register
add R3, R3, R2
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...

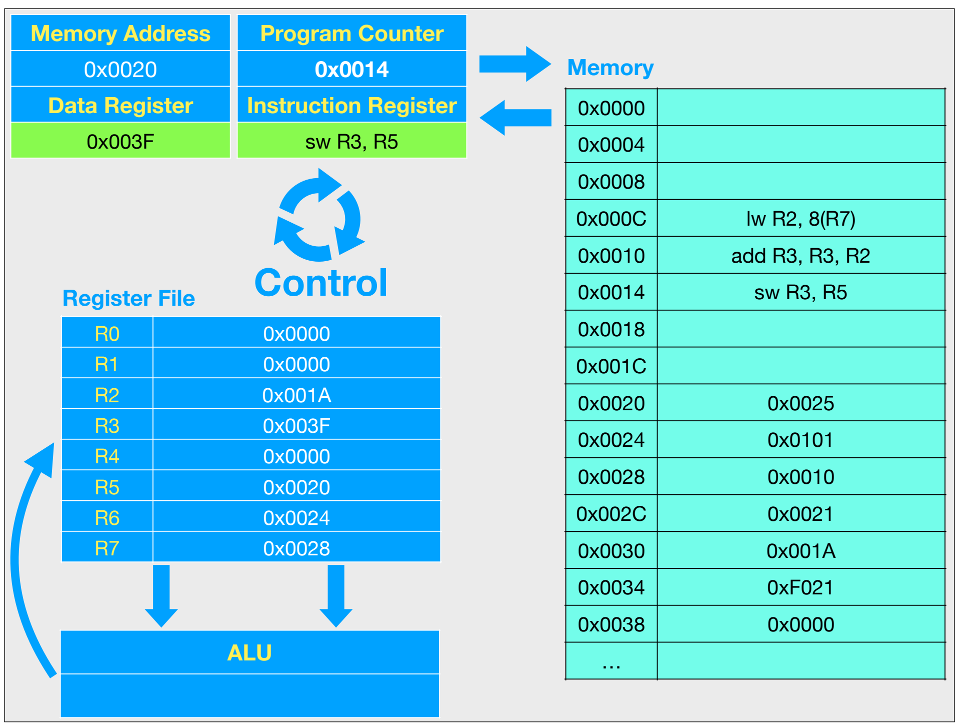

Memory Address
0x0030
Program Counter
0x0014
Data Register
0x001A
Instruction Register
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...

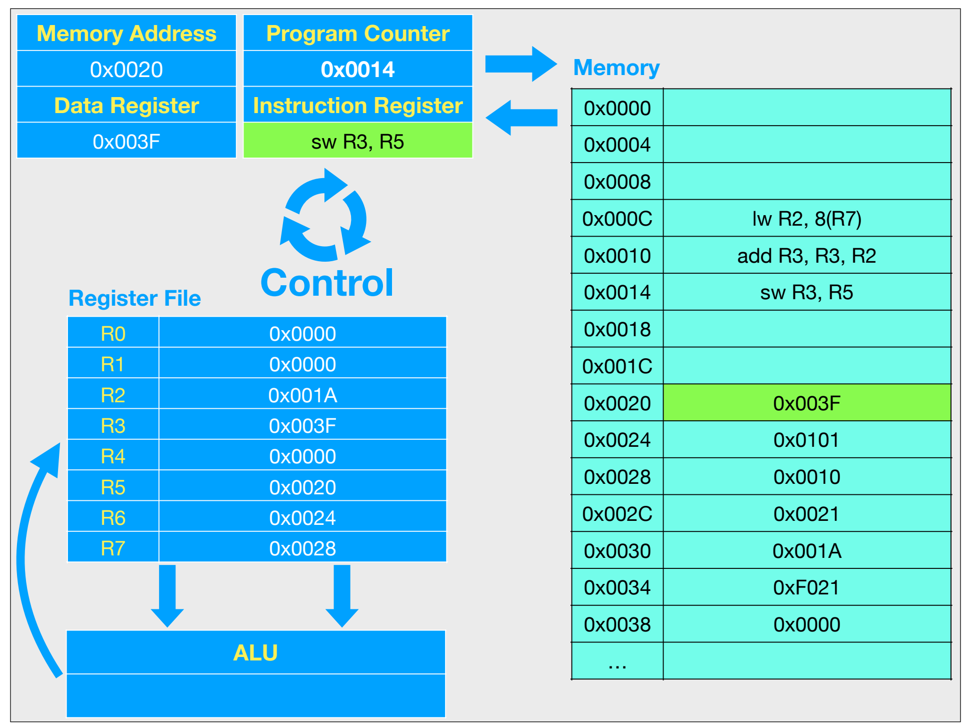

Memory Address
0x0030
Program Counter
0x0014
Data Register
0x001A
Instruction Register
sw R3, R5
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...


Memory Address
0x0030
Program Counter
0x0014
Data Register
0x001A
Instruction Register
sw R3, R5
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
pass trough
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...


Memory Address
0x0020
Program Counter
0x0014
Data Register
0x001A
Instruction Register
sw R3, R5
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...


Memory Address
0x0020
Program Counter
0x0014
Data Register
0x001A
Instruction Register
sw R3, R5
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...


Memory Address
0x0020
Program Counter
0x0014
Data Register
0x003F
Instruction Register
sw R3, R5
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x0025
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...


Memory Address
0x0020
Program Counter
0x0014
Data Register
0x003F
Instruction Register
sw R3, R5
Control
Register File
R0 0x0000
R1 0x0000
R2 0x001A
R3 0x003F
R4 0x0000
R5 0x0020
R6 0x0024
R7 0x0028
ALU
Memory
0x0000
0x0004
0x0008
0x000C lw R2, 8(R7)
0x0010 add R3, R3, R2
0x0014 sw R3, R5
0x0018
0x001C
0x0020 0x003F
0x0024 0x0101
0x0028 0x0010
0x002C 0x0021
0x0030 0x001A
0x0034 0xF021
0x0038 0x0000
...

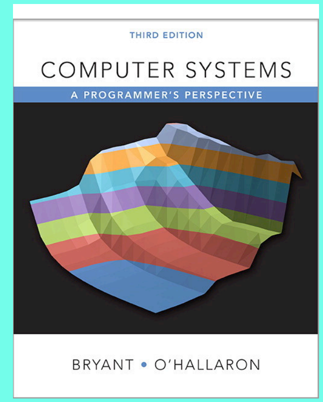
Representing integers

Disclaimer

The following slides are from:

Computer Systems (Bryant and O'Hallaron)

A Programmer's Perspective



Today: Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- **Integers**
  - **Representation: unsigned and signed**
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings
- Summary

Encoding Integers

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

```
short int x =  15213;
short int y = -15213;
```

- **C does not mandate using two's complement**
  - But, most machines do, and we will assume so
- **C `short` 2 bytes long**

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

- **Sign Bit**
  - For 2's complement, most significant bit indicates sign
    - 0 for nonnegative
    - 1 for negative

Two-complement: Simple Example

	-16	8	4	2	1	
10 =	0	1	0	1	0	8+2 = 10

	-16	8	4	2	1	
-10 =	1	0	1	1	0	-16+4+2 = -10

Two-complement Encoding Example (Cont.)

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		**15213**		**-15213**

Numeric Ranges

- **Unsigned Values**
  - *UMin* = 0
    000…0
  - *UMax* = $2^w - 1$
    111…1
- **Two's Complement Values**
  - *TMin* = -2^{w-1}
    100…0
  - *TMax* = $2^{w-1} - 1$
    011…1
  - Minus 1
    111…1

Values for *W* = 16

	Decimal	Hex	Binary
`UMax`	65535	`FF FF`	`11111111 11111111`
`TMax`	32767	`7F FF`	`01111111 11111111`
`TMin`	-32768	`80 00`	`10000000 00000000`
`-1`	-1	`FF FF`	`11111111 11111111`
`0`	0	`00 00`	`00000000 00000000`

Values for Different Word Sizes

	W			
	8	**16**	**32**	**64**
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

- **Observations**
  - |*TMin* | = *TMax* + 1
    - Asymmetric range
  - *UMax* = 2 * *TMax* + 1
  - Question: abs(TMin)?
- **C Programming**
  - #include <limits.h>
  - Declares constants, e.g.,
    - ULONG_MAX
    - LONG_MAX
    - LONG_MIN
  - Values platform specific

Unsigned & Signed Numeric Values

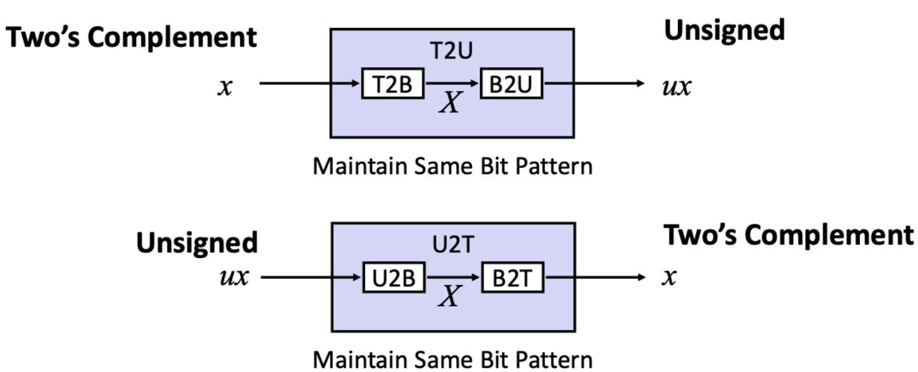
X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	–8
1001	9	–7
1010	10	–6
1011	11	–5
1100	12	–4
1101	13	–3
1110	14	–2
1111	15	–1

- **Equivalence**
  - Same encodings for nonnegative values
- **Uniqueness**
  - Every bit pattern represents unique integer value
  - Each representable integer has unique bit encoding
- **⇒ Can Invert Mappings**
  - U2B(x) = B2U^{-1}(x)
    - Bit pattern for unsigned integer
  - T2B(x) = B2T^{-1}(x)
    - Bit pattern for two's comp integer

Today: Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- **Integers**
  - Representation: unsigned and signed
  - **Conversion, casting**
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings

Mapping Between Signed & Unsigned



- **Mappings between unsigned and two's complement numbers: Keep bit representations and reinterpret**

Mapping Signed ↔ Unsigned

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

→ T2U →

← U2T ←

Mapping Signed ↔ Unsigned

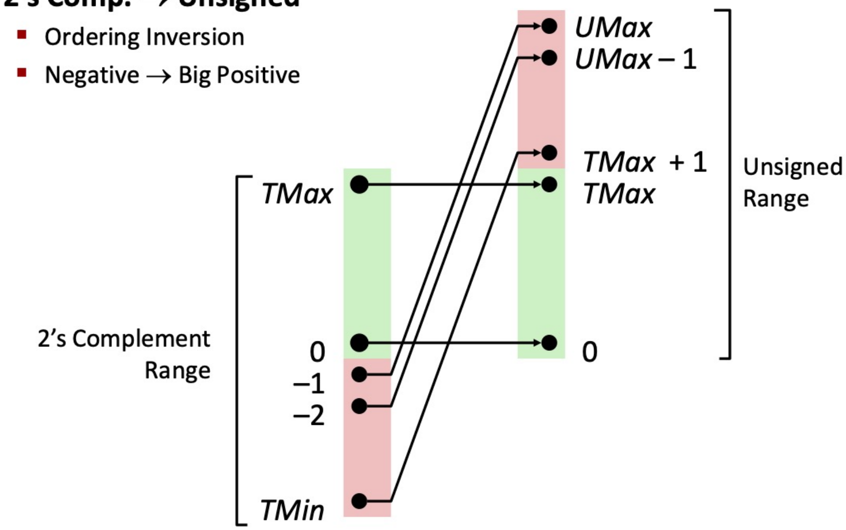
Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

=

+/- 16

Conversion Visualized

- **2's Comp. → Unsigned**
  - Ordering Inversion
  - Negative → Big Positive



Signed vs. Unsigned in C

- **Constants**
  - By default are considered to be signed
  - Unsigned if have "U" as suffix, or if too big to be signed
    `0U, 2147483648`
  - Watch out! A leading minus sign is not part of the constant!
    `-2147483648 == 2147483648` on 32-bit machines (why?)
- **Casting**
  - Explicit casting between signed & unsigned same as U2T and T2U

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```

  - Implicit casting also occurs via assignments and procedure calls

```
tx = ux;                    int fun(unsigned u);
uy = ty;                    uy = fun(tx);
```


Casting Surprises

- **Expression Evaluation**
  - If there is a mix of unsigned and signed in single expression, ***signed values implicitly cast to unsigned***
  - Including comparison operations **<, >, ==, <=, >=**
  - Examples for $W = 32$: **TMIN = -2,147,483,648 , TMAX = 2,147,483,647**

Constant$_1$	Constant$_2$	Relation	Evaluation
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483647-1	>	signed
2147483647U	-2147483647-1	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned
2147483647	2147483648U	<	unsigned
2147483647	(int) 2147483648U	>	signed

Summary
Casting Signed ↔ Unsigned: Basic Rules

- Bit pattern is maintained
- But reinterpreted
- Can have unexpected effects: adding or subtracting 2^w

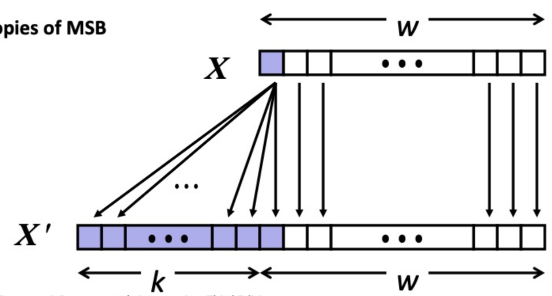
- **Expression containing signed and unsigned int**
  - `int` is cast to `unsigned`!!

Today: Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- **Integers**
  - Representation: unsigned and signed
  - Conversion, casting
  - **Expanding, truncating**
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings

Sign Extension

- **Task:**
  - Given w-bit signed integer x
  - Convert it to $w+k$-bit integer with same value
- **Rule:**
  - Make k copies of sign bit:
  - $X' = \underbrace{x_{w-1}, \ldots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2}, \ldots, x_0$



Sign Extension: Simple Example

Positive number

	-16	8	4	2	1
10 =	0	1	0	1	0

	-32	16	8	4	2	1
10 =	0	0	1	0	1	0

Negative number

	-16	8	4	2	1
-10 =	1	0	1	1	0

	-32	16	8	4	2	1
-10 =	1	1	0	1	1	0

Larger Sign Extension Example

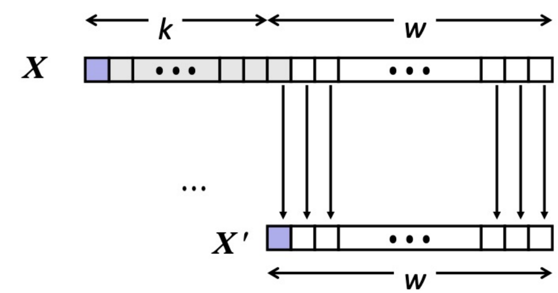
```
short int x =  15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

- **Converting from smaller to larger integer data type**
- **C automatically performs sign extension**

Truncation

- **Task:**
  - Given k+*w*-bit signed or unsigned integer *X*
  - Convert it to *w*-bit integer X' with same value for "small enough" X
- **Rule:**
  - Drop top *k* bits:
  - $X' = x_{w-1}, x_{w-2}, \ldots, x_0$



Truncation: Simple Example

No sign change

	-16	8	4	2	1
2 =	0	0	0	1	0

	-8	4	2	1
2 =	0	0	1	0

`2 mod 16 = 2`

	-16	8	4	2	1
-6 =	1	1	0	1	0

	-8	4	2	1
-6 =	1	0	1	0

`-6 mod 16 = 26U mod 16 = 10U = -6`

Sign change

	-16	8	4	2	1
10 =	0	1	0	1	0

	-8	4	2	1
-6 =	1	0	1	0

`10 mod 16 = 10U mod 16 = 10U = -6`

	-16	8	4	2	1
-10 =	1	0	1	1	0

	-8	4	2	1
6 =	0	1	1	0

`-10 mod 16 = 22U mod 16 = 6U = 6`

Summary:
Expanding, Truncating: Basic Rules

- **Expanding (e.g., short int to int)**
  - Unsigned: zeros added
  - Signed: sign extension
  - Both yield expected result
- **Truncating (e.g., unsigned to unsigned short)**
  - Unsigned/signed: bits are truncated
  - Result reinterpreted
  - Unsigned: mod operation
  - Signed: similar to mod
  - For small (in magnitude) numbers yields expected behavior