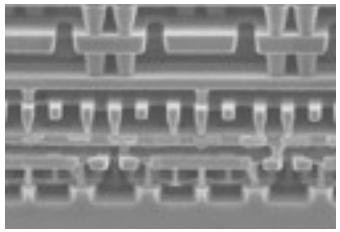
Introducció (1):

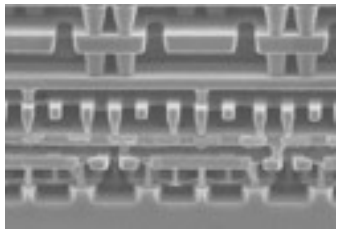
Repàs

What is Computer Systems?



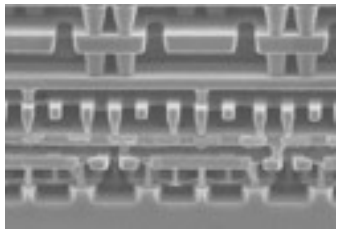
Problem	Who scores the highest on the exam?
Algorithm	Quicksort
Program	Human-readable language (Java, C)
Instruction Set Architecture	Machine Language
Microarchitecture	Hardware Design
Circuit	Electrons, Resistors, Capacitors, etc.

What is Computer Systems?



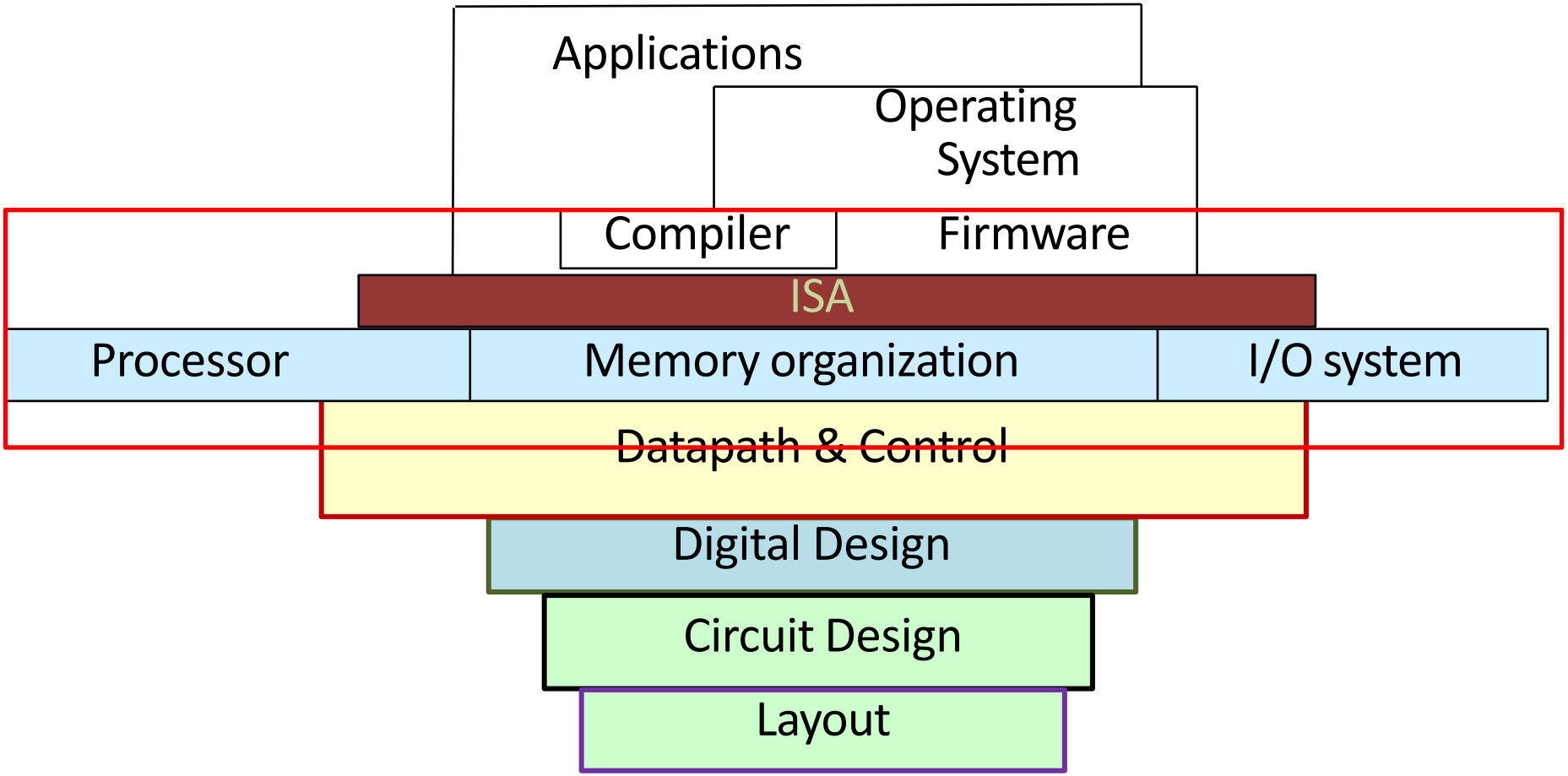
Problem	Who scores the highest on the exam?
Algorithm	Quicksort
Program	Human-readable language (Java, C)
Instruction Set Architecture	Machine Language
Microarchitecture	Hardware Design
Circuit	Electrons, Resistors, Capacitors, etc.

Computer Systems Match User Requirements to Hardware Technologies



Problem	Who scores the highest on the exam?
Algorithm	Quicksort
Program	Human-readable language (Java, C)
Instruction Set Architecture	Machine Language
Microarchitecture	Hardware Design
Circuit	Electrons, Resistors, Capacitors, etc.

Computer System View



Two Fundamental Aspects of Computer Systems

- How is a human-readable program translated to a representation that computers can understand?
- How does a modern computer execute that program?

Problem	Who scores the highest on the exam?
Algorithm	Quicksort
Program	Human-readable language (Java, C)
Instruction Set Architecture	Machine Language
Microarchitecture	Hardware Design
Circuit	Electrons, Resistors, Capacitors, etc.

The "Translation" Process, a.k.a., Compilation

- It translates a text file to an executable binary file (a.k.a., executable) consisting of a sequence of instructions
- Why binary? Computers understand only 0s and 1s

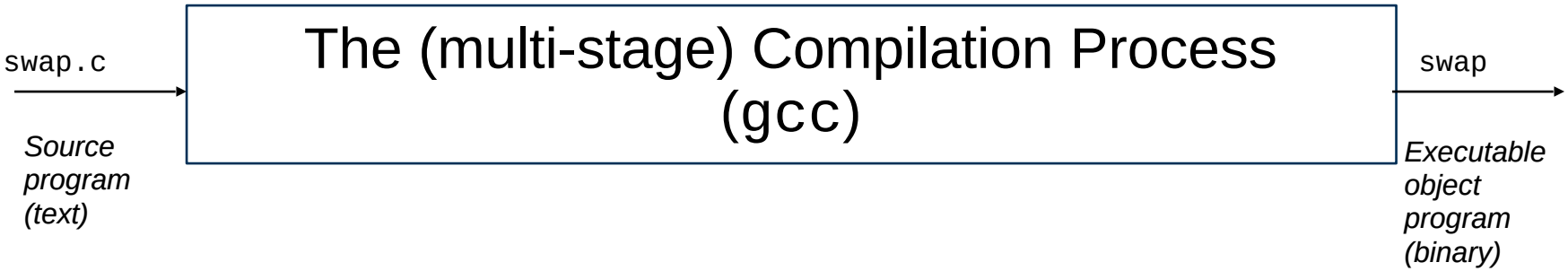
Example: swap.c **(Human readable)**

```
swap(size_t v[], size_t k)
{
size_t temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
printf("%d\n",temp)
}
```

Executable Binary (Machine-readable)

```
00000000011010110010011000100011
00000000011001010000011001100011
00000000000000110011001010000011
00000000100000110011001110000011
00000000111001100110000001000011
00000000101001100110100001000011
00000000000000001000000011001111
```

The “Translation” Process



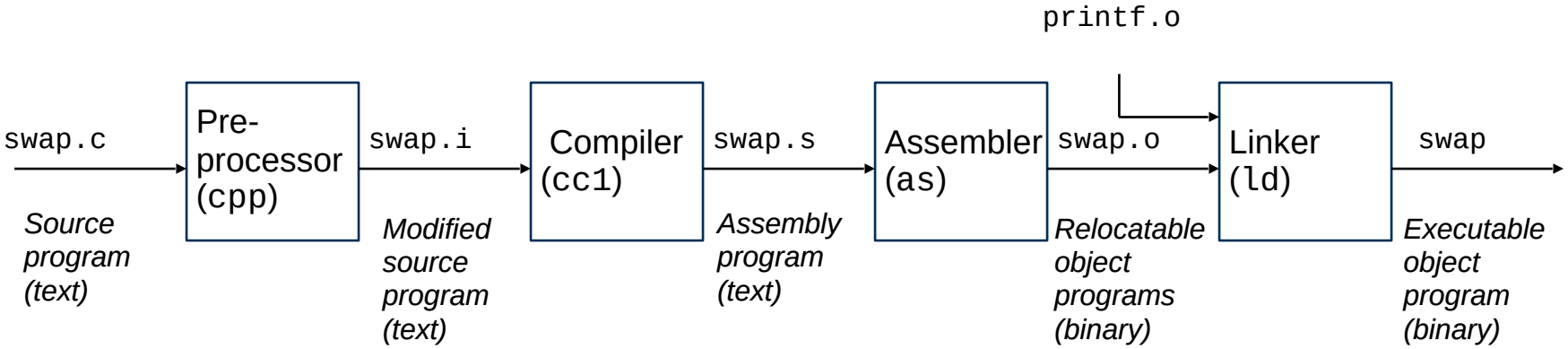
Example: swap.c **(Human readable)**

```
swap(size_t v[], size_t k)
{
size_t temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
printf(“%d\n”,temp)
}
```

Executable Binary (Machine-readable)

```
00000000011010110010011000100111
00000000110010100000011001100111
00000000000000110011001010000011
00000000100000110011001110000011
00000000111001100110000001000111
00000000101001100110100001000111
00000000000000001000000011001111
```

The “Translation” Process



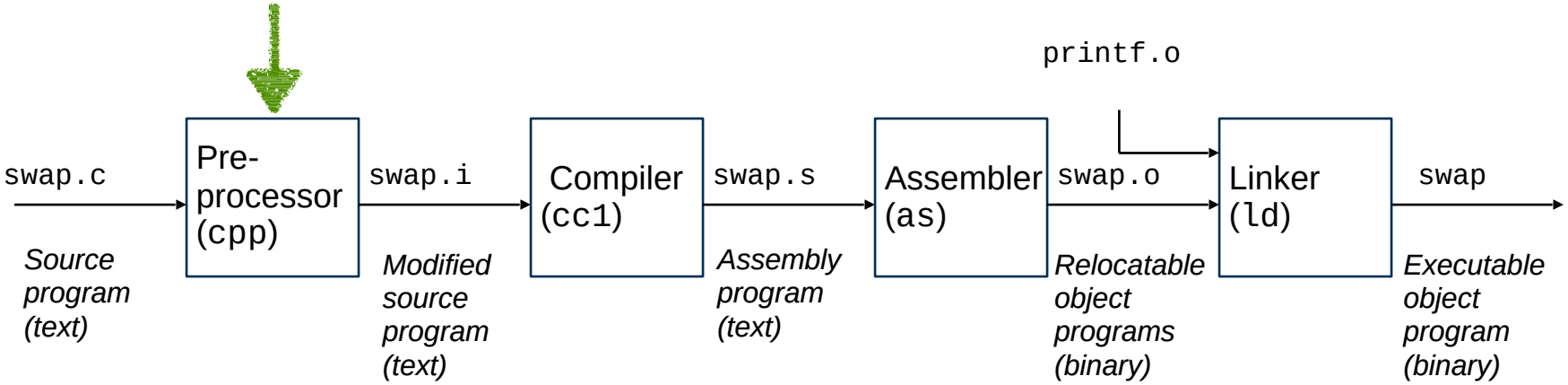
Example: swap.c **(Human readable)**

```
swap(size_t v[], size_t k)
{
size_t temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
printf(“%d\n”,temp)
}
```

Executable Binary (Machine-readable)

```
00000000011010110010011000100111
00000000011001010000011001100111
00000000000000110011001010000011
00000000100000110011001110000011
00000000111001100110000001000111
00000000101001100110100001000111
00000000000000001000000011001111
```

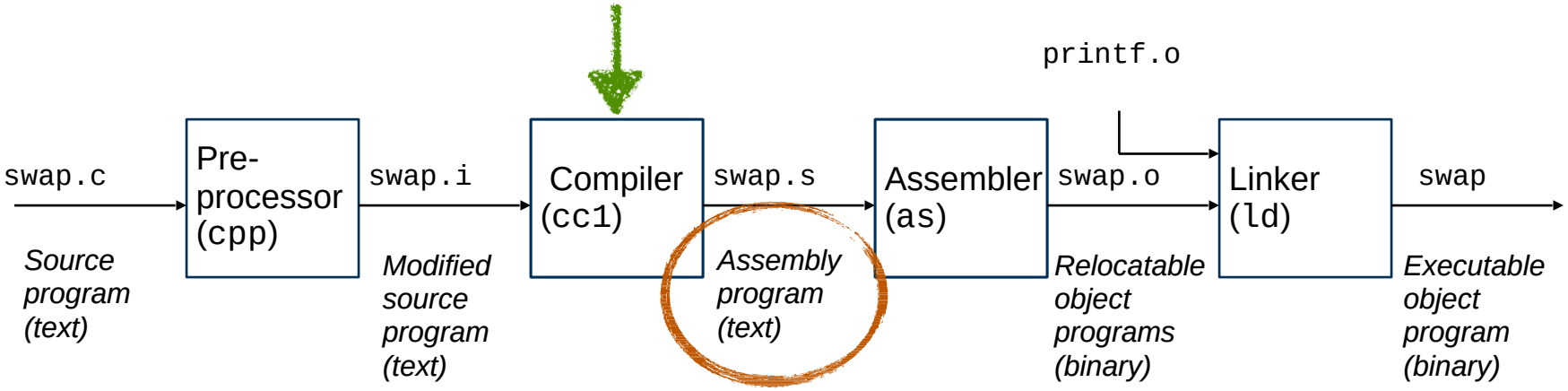
The “Translation” Process



Example: swap.c **(Human readable)**

```
swap(size_t v[], size_t k)
{
size_t temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
printf(“%d\n”,temp)
}
```

The “Translation” Process



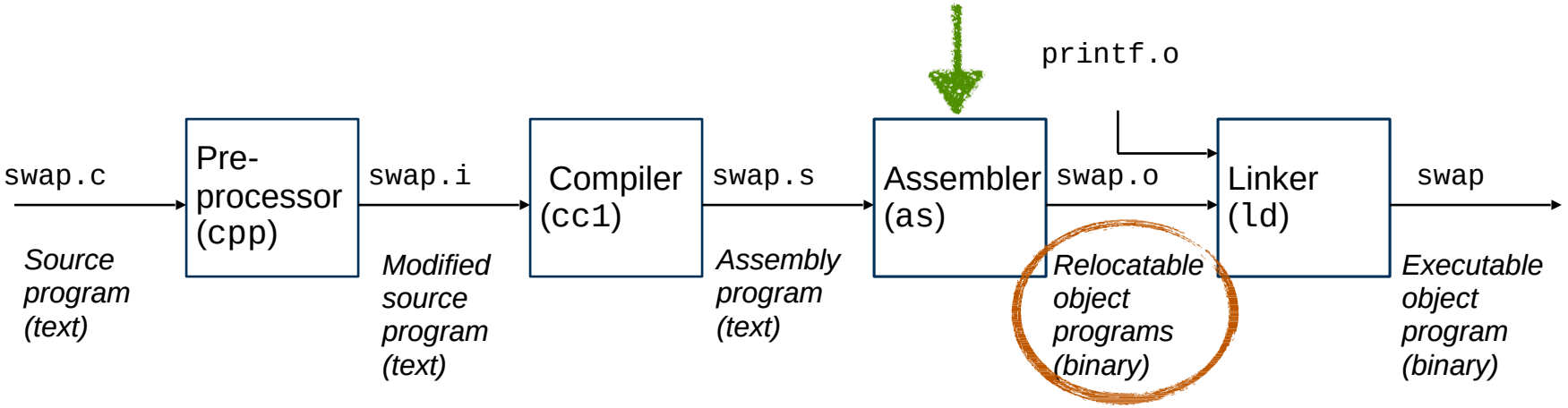
Example: swap.c **(Human readable)**

```
swap(size_t v[], size_t k)
{
size_t temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
printf(“%d\n”,temp)
}
```

Assembly program: swap.s

```
swap:
        slli x6, x11, 3
        add x6, x10, x6
        ld x5, 0(x6)
        ld x7, 8(x6)
        sd x7, 0(x6)
        sd x5, 8(x6)
        jalr x0, 0(x1)
        ecall _printf
```

The "Translation" Process



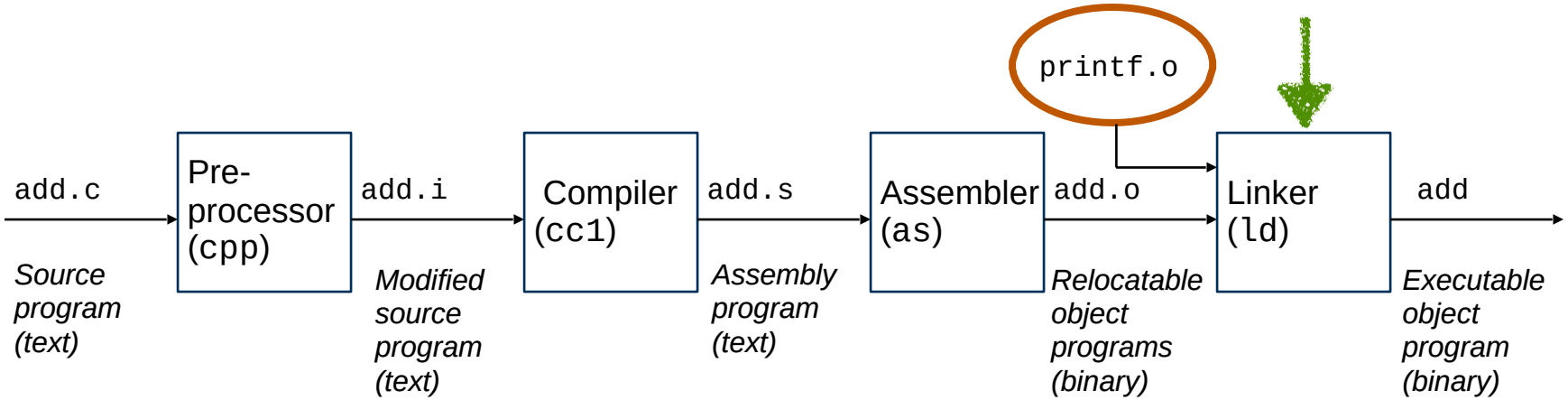
Assembly program: swap.s

```
swap:
        slli x6, x11, 3
        add x6, x10, x6
        ld x5, 0(x6)
        ld x7, 8(x6)
        sd x7, 0(x6)
        sd x5, 8(x6)
        jalr x0, 0(x1)
        ecall _printf
```

Relocatable Binary

```
00000000001101011001001100010011
00000000011001010000011100110011
00000000000000011001100101000011
00000001000001100110011110000011
00000000111001100110000000100011
00000000101001100110100000100011
00000000000000010000000001100111
stub _printf
```

The "Translation" Process



Relocatable Binary

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000011001100101000011
00000001000001100110011100000011
00000000111001100110000000100011
00000000101001100110100000100011
00000000000000001000000001100111
stub _printf
```

Relocatable Binary for `printf.o`

Executable Binary

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000011001100101000011
00000001000001100110011100000011
00000000111001100110000000100011
00000000101001100110100000100011
00000000000000001000000001100111
10100000011100001000000001100111
```

Back to Layers of Transformation…

- How is a human- readable program translated to a representation that computers can understand?

Problem	Who scores the highest on the exam?
Algorithm	Quicksort
Program	Human-readable language (Java, C)
Instruction Set Architecture	Machine Language
Microarchitecture	Hardware Design
Circuit	Electrons, Resistors, Capacitors, etc.

Instruction Set Architecture (ISA)

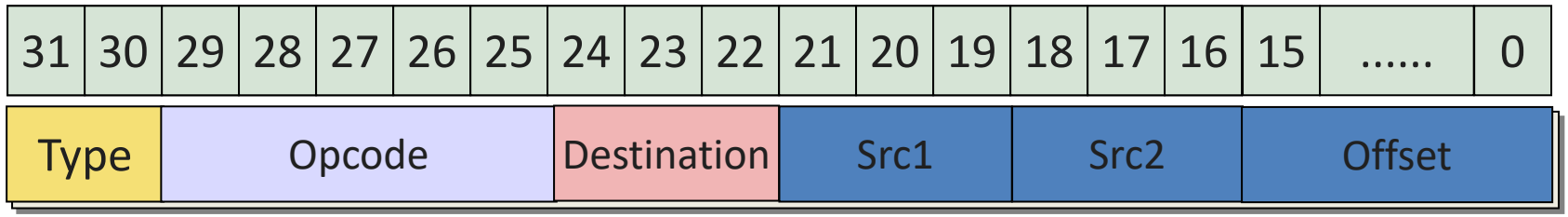
- Instructions are the language the computer understand
- Instruction Set is the vocabulary of that language
- It serves as the hardware/software interface
  - Defines instruction format (bit encoding)
    - Number of explicit operands per instruction
    - Operand location
    - Number of bits per instruction
    - Instruction length: fixed, short, long, or variable., …
  - Examples: *MIPS, Alpha, x86, IBM 360, VAX, ARM, JVM*

Instruction Set Architecture

- Little research on ISA, much more microarch. research
  - ISA is stable now. "One ISA rules them all."
  - Free, open ISA: RISC V (UC Berkeley, https://riscv.org/)

  - Instead, focus on optimizing the implementation.
- Interesting question: can we have one microarchitecture (implementation) for different ISAs?
  - Can a microarchitecture designed for ISA *X* execute ISA *Y*?
  - Yes but you need something that translates programs written in ISA *Y* to ISA *X* while you are executing it: *dynamic binary translator*
  - E.g., Transmeta executes x86 ISA programs on their in-house ISA

Register Instructions



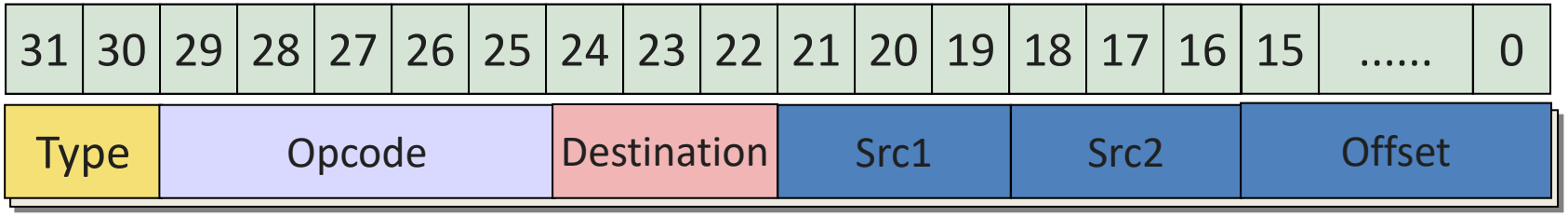
Arith.
Logic
Move
Shift

Instrucció	acció
Op Dest src1 src2	CR[Dest.]<=CR[src1] Op CR[src2]

Example: R2 = R0 + R1 =>
Type = Reg. Inst => 2'b00
Opcode = Sum => 5'b00010
Destination reg = R2 => 3'b010
Src1 = R0 => 3'b000
Src2 = R1 => 3'b001
Offset = don't care

Instruction:
00_00010_010_000_001_0000000000000000

Memory Instructions



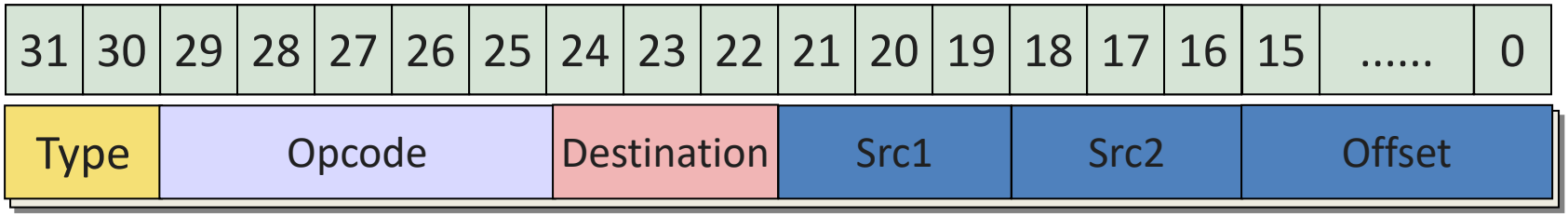
Load
Store

Instrucció	acció
L immH Dest.	CR[Dest.][31..16]<=Offset
L immL Dest.	CR[Dest.][15..0]<=Offset
L rel Dest. src2 Offset	CR[Dest.]<= Mem[CR[src2]+Offset]
S rel src1 src2 Offset	Mem[CR[src2]+Offset] <= CR[src1]

Example: Load R2, R0, Offset =>
Type = Reg. Inst => 2'b01
Opcode = Sum => 5'b00011
Destination reg = R2 => 3'b010
Src1 = don't care
Src2 = R1 => 3'b001
Offset = 16'b0010010000010000

Instruction:
01_00011_010_000_001_0010010000010000

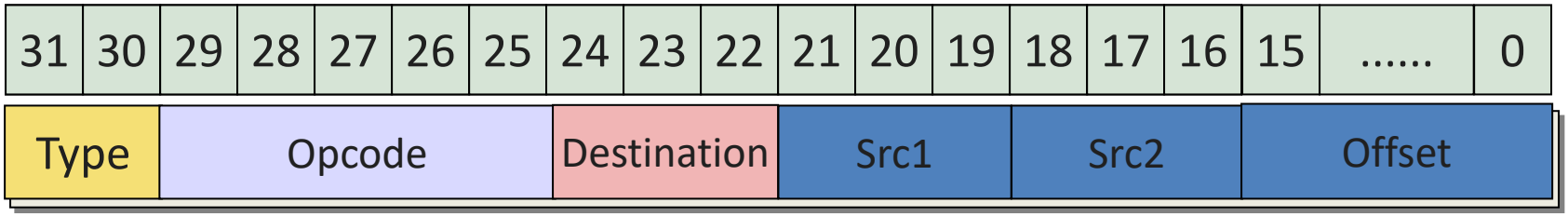
Control Instructions



Jump
Branch

Instrucció	acció
Jump Offset	PC <= PC + Offset
Jump src2 Offset	PC <= CR[src2] + Offset
Brel Offset	PC <= PC+1 if CR[src1] notrel CR[src2]
	PC<=PC+Offset if CR[src1] rel CR[src2]

Other Instructions



No-op
Clear
Set
Reset

Instrucció	acció
No-op	No Operation
Clear Dest.	CR[Dest.] <= 0
Sstat Dest.	Status[Dest.] <= 1
Rstat Dest.	Status[Dest.] <= 0

Exemple de problema

1. Tenim un processador amb 6 instruccions. La codificació de les 6 instruccions es mostren a la taula següent:

Instruction	Opcode	16-bit encoding				Function
Mov Ra, d	0000	Opcode (4 bits)	Destination Register (4 bits)	Address (8 bits)		**RF[a] <-M[d]**
Mov d, Ra	0001	Opcode (4 bits)	Source Register (4 bits)	Address (8 bits)		**M[d] <-RF[a]**
Add Ra,Rb,Rc	0010	Opcode (4 bits)	Destination Register (4 bits)	Source Register (4 bits)	Source Register (4 bits)	**RF[a] <-RF[b] + RF[c]**
Mov Ra, #C	0011	Opcode (4 bits)	Destination Register (4 bits)	Constant (8 bits)		**RF[a] <- c**
Sub Ra,Rb,Rc	0100	Opcode (4 bits)	Destination Register (4 bits)	Source Register (4 bits)	Source Register (4 bits)	**RF[a] <-RF[b] - RF[c]**
Jumpz Ra, X	0101	Opcode (4 bits)	Source Register (4 bits)	Offset (8 bits)		**If RF[a] == 0, PC<-PC+offset**

Volem ampliar el nombre d'instruccions a 26 (afegim 20 instruccions). Aquestes instruccions són úniques (exemple: Mult, And, Or, etc.). Hem de mantenir el tipus d'instrucció amb la codificació de 16 bits. Com afectarà l'addició de 20 instruccions a la instrucció "Add" ? (*altres instruccions poden ser afectades, tot i així la qüestió només pregunta sobre la instrucció "Add")*

Back to Layers of Transformation…

- How is a human- readable program translated to a representation that computers can understand?
- How does a modern computer execute that program?

Problem	Who scores the highest on the exam?
Algorithm	Quicksort
Program	Human-readable language (Java, C)
Instruction Set Architecture	Machine Language
Microarchitecture	Hardware Design
Circuit	Electrons, Resistors, Capacitors, etc.

The Fundamental Idea of Computers

- Executables (i.e., instructions) are stored in "memory"
- Processors read instructions from memory and execute instructions one after another

Assembly program: swap.s

```
swap:
        slli x6, x11, 3
        add x6, x10, x6
        ld x5, 0(x6)
        ld x7, 8(x6)
        sd x7, 0(x6)
        sd x5, 8(x6)
        jalr x0, 0(x1)
```

High-level Organization of Computer Hardware a.k.a., The Von Neumann Model