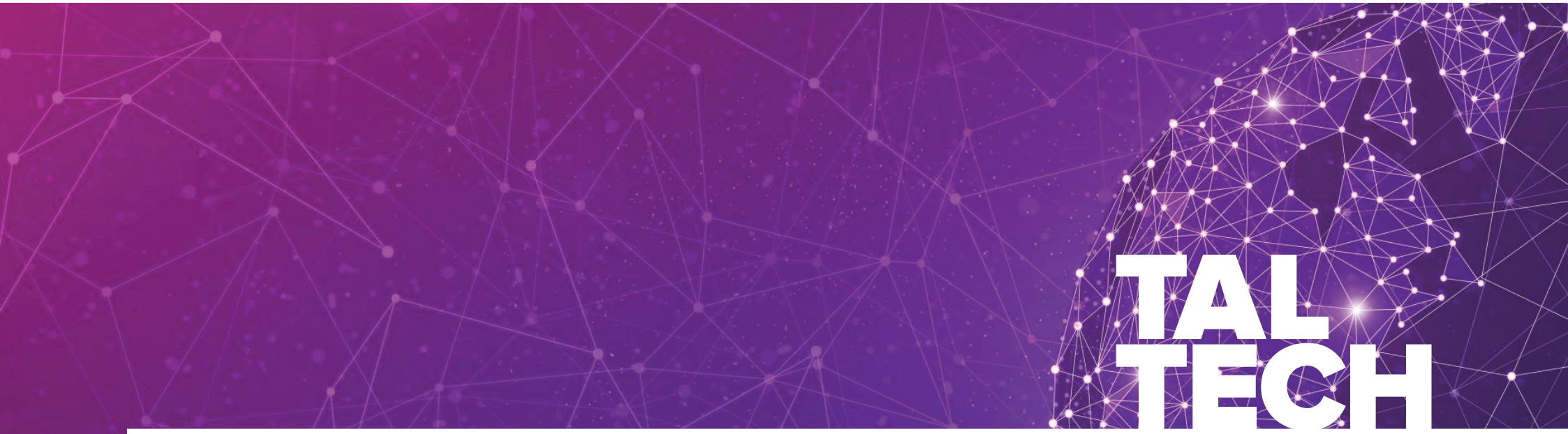


MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

17.09.2021

THE ISA - 1

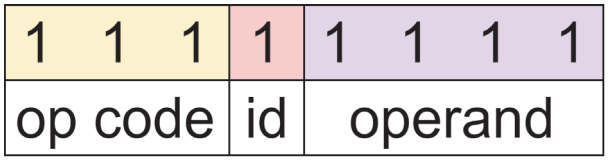
- **What is the Instruction Set Architecture?**
  - The ISA is the **implementation architecture** of the computer.
  - It **dictates everything inside the computer processor, memory, and even what external devices can connect to it.**
  - It is the **single most important factor in designing a computer**.
    - It decides how the CPU is built.
    - It defines the functions the CPU is capable to perform
    - It defines what types of data the CPU can process.
  - The importance of the ISA comes from it being the **vocabulary of the CPU**.
    - The CPU **internal circuitry is built to translate the instruction** into functions and operations.
    - Since the CPU internal circuitry is based on the instructions, the **ISA becomes the only thing that the CPU understands**.
  - The ISA is **a list of functions and operations that the CPU can perform and describes the type of data and what resources needed to complete these functions and operations.**

THE ISA - 1

- **Believe it or not, we have our own ISA already made!**

THE ISA - 1

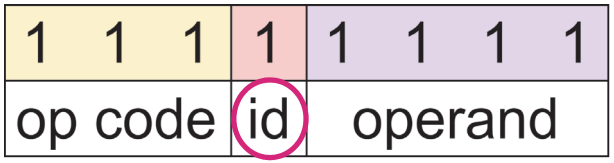
- **Believe it or not, we have our own ISA already made!**
  - We created an ISA last class. A fully functioning ISA for the 8-bit Dummy CPU.



Op code	operation	Function	Use	binary
001	LD	Load to accum	LD # 5	00110101
010	ST	Store to memory	ST $ 2	01000010
011	ADD	Add value to value in accum	ADD # 10	01111010
100	SUB	Subtract value from value in accum	SUB # 4	10010100
101	EQ	Checks in value is equal to value in accum, if true, skip next instruction	EQ # 5	10110101
110	JP	Set value in PC to value	JP $ 8	11001000
111	HE	Halt Execution	HE	11100000

THE ISA - 1

- **Subjects for modifications – how to interpret the id-bit?**
- For instance, "LD $ 5" [00100101] – Load to accum from memory, address in word 5.



Op code	operation	Function	Use	binary
001	LD	Load to accum	LD # 5	00110101
010	ST	Store to memory	ST $ 2	01000010
011	ADD	Add value to value in accum	ADD # 10	01111010
100	SUB	Subtract value from value in accum	SUB # 4	10010100
101	EQ	Checks in value is equal to value in accum, if true, skip next instruction	EQ # 5	10110101
110	JP	Set value in PC to value	JP $ 8	11001000
111	HE	Halt Execution	HE	11100000

THE ISA - 2

- **What are the important things to know in order to make an ISA?**
  - Must keep in mind at all times that **any decision made, becomes a constraint**.
    - E.g. When choosing the size of any part of the ISA, the size is a constraint on other parts since the instruction length is finite.
    - When we choose 3-bits for op code in the Dummy CPU ISA **001**00101

    We limited ourselves to only 5 bits left that we can use for other things.

    This also limited us to only 8 possible operations to implement.
    - **Decisions are double edged swords**.
  - There are infinite designs possible, but the majority are too complex to be practical.
    - Choose simplicity over everything.
    - The simpler the ISA, the easier it is to design a CPU that executes it.
  - **Accept that all designs have flaws**.
  - What types of **resources** is your ISA going to use:
    - How many register? ALU functions? Memory?
    - Any external devices?
  - **Always be prepared to change your ISA later!**

THE ISA - 2

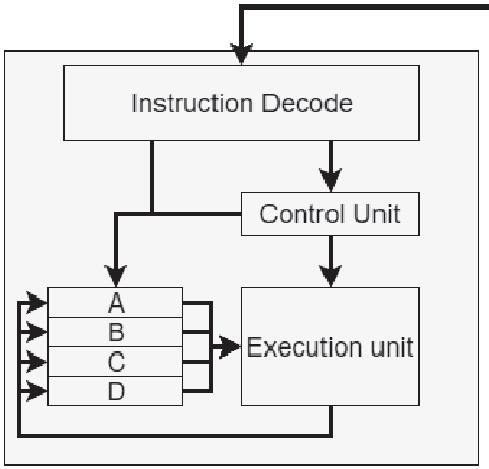
- **What to think about when designing an ISA?**
  - How many **registers** I want to use?
    - Registers are fast and take very small space, but have small capacity as well.
    - **More registers**, means less access to slow memory – Better performance.
    - **Less registers**, means less complexity – better applicability.
  - What **Bus** am I using, how big?
    - The bus carries data, instructions, and control signals. However, an 8 bit ISA needs an 8-bit bus, a 16-bit ISA needs 16-bit bus.
    - The bus takes a large area on chip. **Smaller bus** means more area of other things.
    - But, a **larger bus**, means more bits space for functionality and data transmitted.
  - What **operation** do I need to perform?
    - The ISA must take in consideration how do we intend to use the CPU?
    - What operation does it make sense to implement?
    - What functions those operations must have?
    - What do I operate on?

THE ISA – 2

- **What to think about when designing an ISA?**
  - What types of **operands** will I have?
    - Addresses? Text? Integers? Floating Point numbers?
    - Do I need to create different operations for different operands?
  - Will need different instruction **formats**?
    - How can I divide the instruction in different formats for different operations?
    - Not all operations that seems to need their own format should have their own format!
    - Some operations look like they do not need a format, but are much easier to implement with a format.
  - What machines do I want to implement this ISA on!
    - Single core? Multi core?
    - PC? Game console? Calculator?

THE ISA - 3

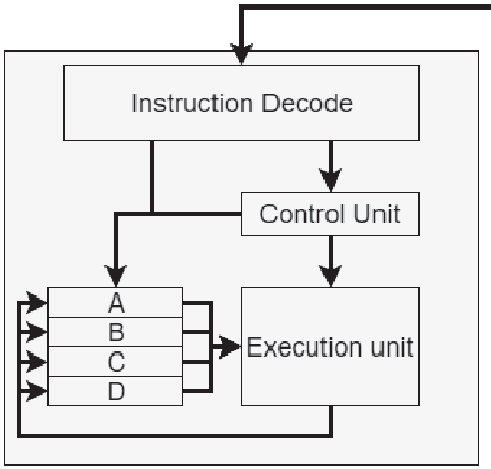
- **There are 2 ISAs we will explore:**
- **RISC:**
  - Stands for **R**educed **I**nstruction **S**et **C**omputer.
  - It is designed to interpret instructions at 1 or 2 cycle at a time.
  - Instructions do one specific thing. Decoding process is simple.
  - RISC is **software** friendly. It gives more room for programmers to be creative. Meaning that most of the work is done on software level.
  - The CPU design is very simple. This allows more on-chip space for resources (reg, mem).
- Lets say we want to multiply 2 and 5 using an operation **PROD**.
  - We will need to load **2** and **5** to registers **A** and **B**
  - We then multiply registers **A** and **B** using **PROD**
  - Then we store value from the product in reg **A** to the memory location **10**
  - This takes 4 instructions.



0	LOAD A, 10
1	LOAD B, 11
2	PROD A, B
3	STORE 10, A
4	
5	
6	
7	
8	
9	
10	2
11	5
12	
13	
14	
15	

THE ISA - 3

- **There are 2 ISAs we will explore:**
- **RISC:**
  - RISC uses a **compiler** that **converts high level languages into instructions.**
  - The RISC compiler does most of the work.
  - This takes a long time, making it a **slow conversion**.
  - Although **programs are simple and straight forward**, those **programs are usually very resource hungry** and might take longer to execute.
  - More instructions, means programs take **more space in memory**.
  - It **needs a collection of general purpose registers to complete operations**.

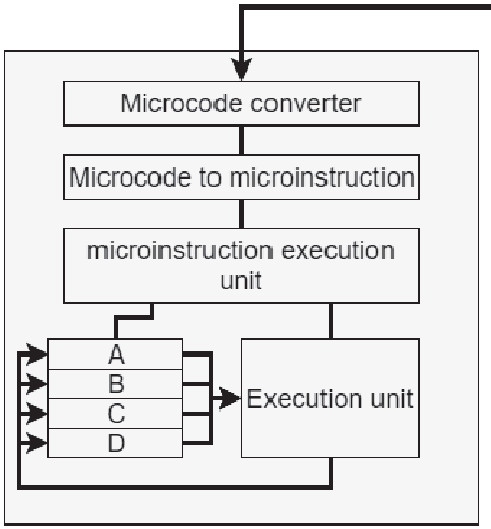


0	LOAD A, 10
1	LOAD B, 11
2	PROD A, B
3	STORE 10, A
4	
5	
6	
7	
8	
9	
10	2
11	5
12	
13	
14	
15	

THE ISA - 3

- **CISC:**
  - Stands for **C**omplex **I**nstruction **S**et **C**omputer.
  - It is designed to complex tasks with the least possible number of instructions.
  - Decoding process is complex and need more space on-chip to be implemented.
  - CISC is **hardware** based. The hardware is designed to "understand" instructions and execute a series of operations using one instruction only.
  - The CPU design is very complex. This allows less on-chip space for resources (reg, mem).

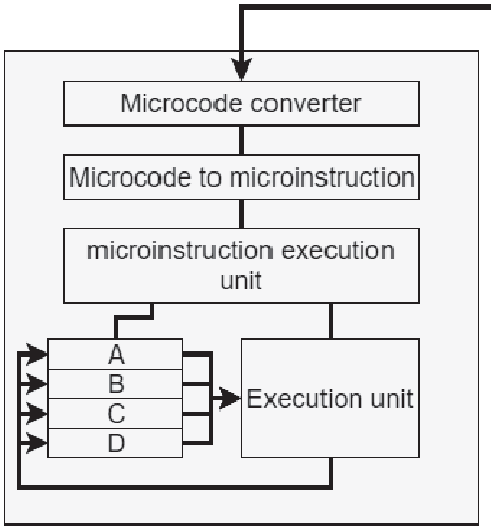
- Lets say we want to multiply 2 and 5 using an operation **MUTL**.
  - We only need to specify the operations **MULT**, then point out to where in memory we are storing 2 and 5.
  - The instruction is then is translated to **microcode** then to **microinstructions**, then it is executed as needed.



0	MULT 10, 11
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	2
11	5
12	
13	
14	
15	

THE ISA - 3

- **CISC:**
  - CISC allows **compilers to do little work** in compiling high level languages.
  - The use of **microcode allows the majority of the work to be done on the hardware level**.
  - Microcode is **then converted in microinstructions**, similar to regular RISC instructions and **executed in a specialized execution unit**.
  - CISC has great advantages when it comes to **complex operations that involve a large number of resources to complete** (mathematical, graphic, etc.)
  - However, as a trade off to that, registers are more likely to be used **per instructions and are refreshed after each instruction is executed**.
  - But, CISC takes **little space in memory as the number of instructions is low**.



0	MULT 10, 11
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	2
11	5
12	
13	
14	
15	

	RISC		CISC	
Attribute	Evaluation	Reason	Evaluation	Reason
Instruction decoding	Low	Instructions are hardcoded according to the logic used for designing the CPU.	Complex	Instructions are implemented as functions not as logical operation. Those functions are then executed using a complex internal system.
Registers	Higher number	Simple CPU design take small area of the chip, which allows more area for on-chip registers and on-chip memory.	Lower number	Complex CPU design leaves little room for on-chip registers.
Cycles per instruction	One/two cycles per instruction	Instructions are simple and easy to execute. Some require longer time, but cycle times are uniform.	Different cycle times	Instructions are complex and do multiple tasks
Memory	Needs more memory	More instructions are needed to do complex tasks, meaning that more space is required to store these instructions.	Needs less memory	One instruction does multiple tasks which reduces the total number of instructions to be stored in memory.
Memory access	Simple	Limited number of memory access modes.	Complex	Has much more ways of accessing memory, especially indexed accesses.
Instruction complexity	Low	Hardware is built to accommodate instructions	High	Instructions are designed to accommodate hardware
Pipelining	Possible	Because the number of cycles needed for an instruction can be easily divided into stages of execution.	Hard	Instructions have no uniform time of execution, making pipelining hard and almost impossible.
Operational work	Less	Registers hold on to information after the execution of an instruction is finished. This information is then used by other instructions	More	Registers are emptied after each instruction, requiring all data to be written into memory and reloaded to registers multiple times.
Clocks	Single clock	All parts of the CPU operate on the same clock cycle	Multi-clock	The complexity of the CPU design requires different parts of the CPU to operate on different frequencies.
Design Target	Reduce the number of cycles per instruction at the cost of total number of instructions needed to execute a program. Less cycles and more memory		Reduce the number of instructions needed to execute a program at the cost of having different cycles per each instruction. More cycles and less memory.	

THE ISA – 4

- **Performance Evaluation**
  - Performance evaluation is sued to measure the time needed for a program to finish executing.
    - What do we need to account for when we measure time per program?

THE ISA – 4

- **Performance Evaluation**
  - Performance evaluation is sued to measure the time needed for a program to finish executing.
    - What do we need to account for when we measure time per program?
      - Time needed to finish a cycle (**T**ime/**C**ycle) or (T/C)
      - The number of cycles per instruction (**C**ycle / **I**nstruction) or (C/I)
      - The number of instructions per program (**I**nstruction / **P**rogram) or (I/P)

$$\frac{time}{program} = \frac{time}{cycle} \times \frac{cycles}{instruction} \times \frac{instructions}{program}$$

THE ISA – 4

- **Performance Evaluation**
  - Performance evaluation is sued to measure the time needed for a program to finish executing.
    - What do we need to account for when we measure time per program?
      - Time needed to finish a cycle (**T**ime/**C**ycle) or (T/C)
      - The number of cycles per instruction (**C**ycle / **I**nstruction) or (C/I)
      - The number of instructions per program (**I**nstruction / **P**rogram) or (I/P)

$$\frac{time}{program} = \frac{time}{cycle} \times \frac{cycles}{instruction} \times \frac{instructions}{program}$$

      - If instructions have different number of cycles, each instruction must be accounted according to its cycle time.

      - **Lets take an example.**

THE ISA - 4

- Lets take the following program and table:

$$\frac{time}{program} = \frac{time}{cycle} \times \frac{cycles}{instruction} \times \frac{instructions}{program}$$

- A cycle takes 1.2 ns to complete.

0	LD $0
1	ADD $1
2	EQ #2
3	JP $2

Operation	use	# of cycles
LD	LD #5	1
	LD $5	2
ST	ST #2	1
	ST $2	1
ADD	ADD #10	1
	ADD $10	2
SUB	SUB #4	1
	SUB $4	2
EQ	EQ #5	1
JP	JP $8	1
HE	HE	1

THE ISA - 4

- Lets take the following program and table:

$$\frac{time}{program} = \frac{time}{cycle} \times \frac{cycles}{instruction} \times \frac{instructions}{program}$$

0	LD $0
1	ADD $1
2	EQ #2
3	JP $2

- A cycle takes 1.2 ns to complete.
- We look for the instructions we are that take similar number of cycles
  - LD $0 and ADD $1 take 2 cycles, 2 instructions that take 2 cycles
  - EQ #2 and JP $2 take 1 cycle, 2 instructions that take 1 cycle

Operation	use	# of cycles
LD	LD #5	1
	LD $5	2
ST	ST #2	1
	ST $2	1
ADD	ADD #10	1
	ADD $10	2
SUB	SUB #4	1
	SUB $4	2
EQ	EQ #5	1
JP	JP $8	1
HE	HE	1

THE ISA - 4

- Lets take the following program and table:

$$\frac{time}{program} = \frac{time}{cycle} \times \frac{cycles}{instruction} \times \frac{instructions}{program}$$

0	LD $0
1	ADD $1
2	EQ #2
3	JP $2

Operation	use	# of cycles
LD	LD #5	1
	LD $5	2
ST	ST #2	1
	ST $2	1
ADD	ADD #10	1
	ADD $10	2
SUB	SUB #4	1
	SUB $4	2
EQ	EQ #5	1
JP	JP $8	1
HE	HE	1

- A cycle takes 1.2 ns to complete.
- We look for the instructions we are that take similar number of cycles
  - LD $0 and ADD $1 take 2 cycles, 2 instructions that take 2 cycles
  - EQ #2 and JP $2 take 1 cycle, 2 instructions that take 1 cycle

- Now, we need to calculate the time need for instructions with different cycles
  - Number of instructions that take 2 cycles (C/I) is 2 per program (I/P)
  - Which means that they take: 1.2 ns * 2 * 2 = **4.8 ns**

THE ISA - 4

- Lets take the following program and table:

$$\frac{time}{program} = \frac{time}{cycle} \times \frac{cycles}{instruction} \times \frac{instructions}{program}$$

0	LD $0
1	ADD $1
2	EQ #2
3	JP $2

Operation	use	# of cycles
LD	LD #5	1
	LD $5	2
ST	ST #2	1
	ST $2	1
ADD	ADD #10	1
	ADD $10	2
SUB	SUB #4	1
	SUB $4	2
EQ	EQ #5	1
JP	JP $8	1
HE	HE	1

- A cycle takes 1.2 ns to complete.
- We look for the instructions we are that take similar number of cycles
  - LD $0 and ADD $1 take 2 cycles, 2 instructions that take 2 cycles
  - EQ #2 and JP $2 take 1 cycle, 2 instructions that take 1 cycle

- Now, we need to calculate the time need for instructions with different cycles
  - Number of instructions that take 2 cycles (C/I) is 2 per program (I/P)
  - Which means that they take: 1.2 ns * 2 * 2 = **4.8 ns**
  - Number of instructions that take 1 cycle (C/I) is 2 per program (I/P)
  - Which means that they take: 1.2 ns * 1 * 2 = **2.4 ns**
  - The total timing needed to finish this small program is 4.8 + 2.4 = **7.2 ns**