EPIC
Institute of Technology
Powered by epam

How everything is calculated?

Course: Systems Architecture

Lecturer: Gleb Lobanov

April 6, 2024

EPIC
Institute of Technology
Powered by epam

Contents

EPIC
Institute of Technology
Powered by epam

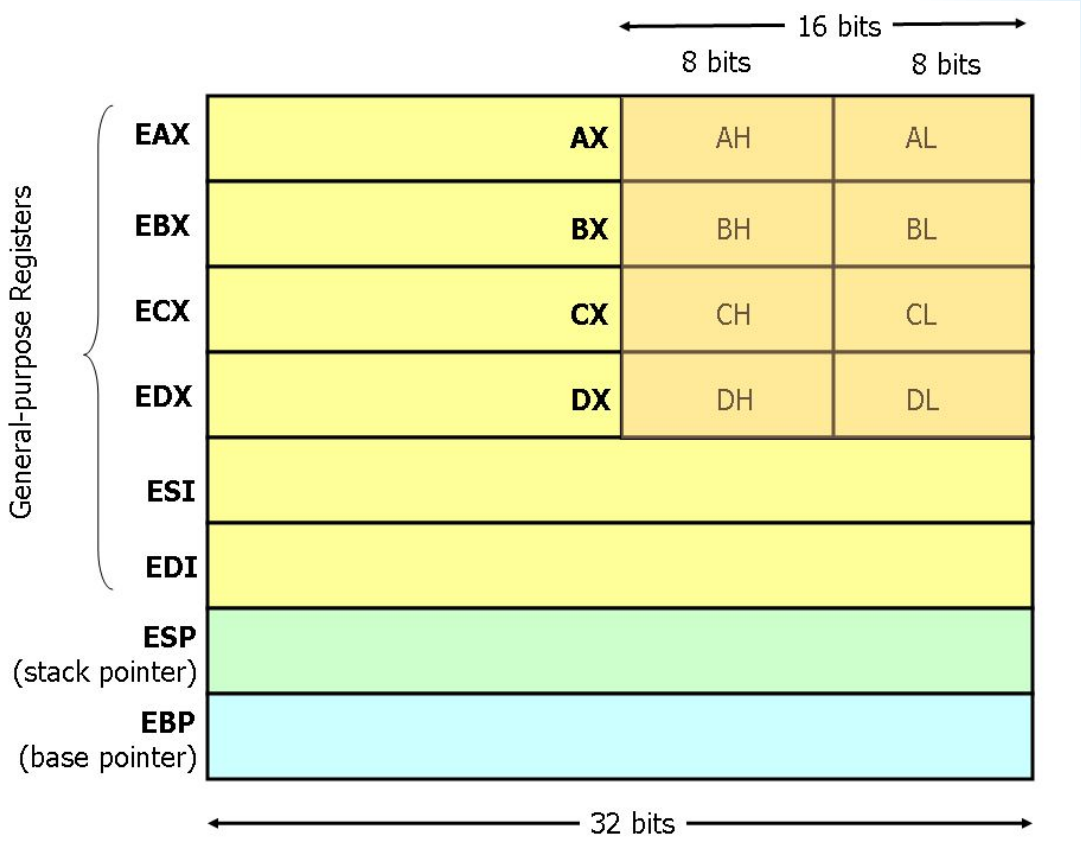
01

Instructions

A little topic about instructions

EPIC
Institute of Technology
Powered by epam

Registers



EPIC Institute of Technology
Powered by epam

RAM model

```
1  ; Calculates sum of R1 - R10 and saves it in R1
2
3  R0 = 5
4
5  loop:
6  R1 = R1 + (R0) ; add
7  R0 = R0 - 1 ; In R0, we store the index of the last non-zero register
8  GGZ R0, loop ;while R0 > 0 we add register to R1 and go to the next element
9
10 R1 = R1 / 2 ; we make R1 = R1 + R1 in the last step
11 HALT
```

+ −

EKM	0
LKM	0
R0	10
R1	3
R2	2
R3	3
R4	2
R5	3

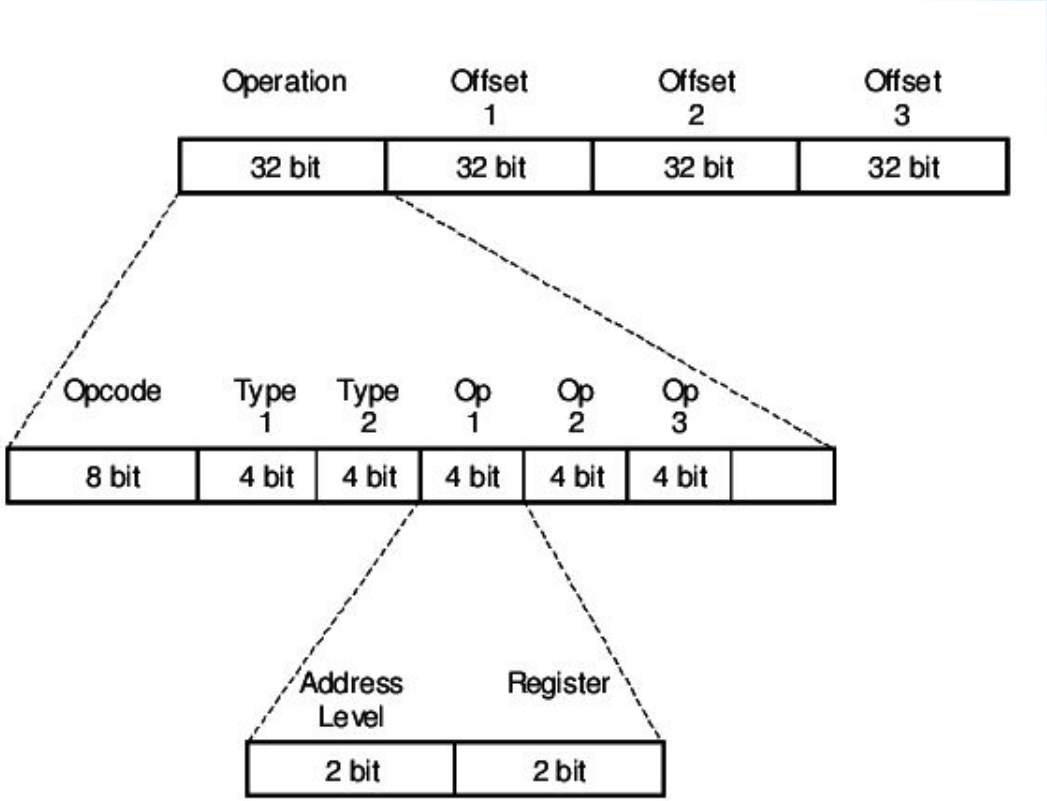
EPIC
Institute of Technology
Powered by epam

Instructions

Normal instructions				Compressed	
00000	SUB	10000	CMP	000	SUB
00001	AND	10001	TEST	001	AND
00010	ADD	10010	LW	010	ADD
00011	OR	10011	SW	011	CMP
00100	XOR	10100	LH	100	LW
00101	LSR	10101	SH	101	SW
00110	LSL	10110	LB	110	LDI
00111	ASR	10111	SB	111	MOV
01000	BREV	11000	LDI	Reserved for FPU	
01001	LDILO	11001			
01010	MPYUHI	Special Insn		11010	FPADD
01011	MPYSHI			11011	FPSUB
01100	MPY	11100	BREAK	11100	FPMPY
01101	MOV	11101	LOCK	11101	FPDIV
01110	DIVU	11110	SIM	11110	FPI2F
01111	DIVS	11111	NOOP	11111	FPF2I

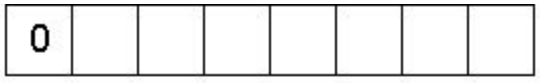
EPIC
Institute of Technology
Powered by epam

Operation



EPIC
Institute of Technology
Powered by epam

Example



Function:
000 ADD
001 – SUB
010 – AND
011 – OR
100 – XOR
101 – INC
110 – DEC
111 – NOT

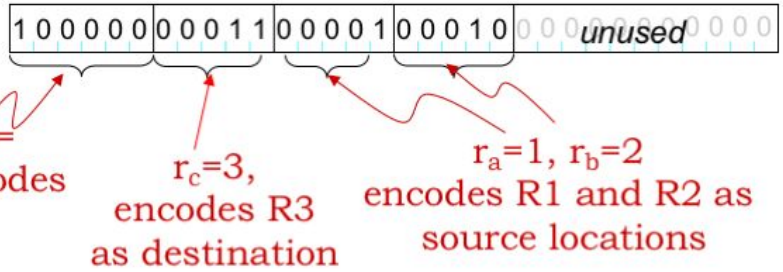
Destination:
00 – Accumulator A
01 – Accumulator B
10 – Index register X
11 – Double accumulator D

Source:
00 – Indirect (X used as pointer)
01 – Accumulator B
10 – Constant*
11 – Memory**

Beta ALU Instructions

Format:

Example coded instruction: ADD



32-bit hex: 0x80611000

We prefer to write a symbolic representation: `ADD(r1,r2,r3)`

```
ADD(ra,rb,rc):
  Reg[rc] ← Reg[ra] + Reg[rb]
```

"Add the contents of ra to the contents of rb; store the result in rc"

Similar instructions for other ALU operations:

- arithmetic: ADD, SUB, MUL, DIV
- compare: CMPEQ, CMPLT, CMPLE
- boolean: AND, OR, XOR, XNOR
- shift: SHL, SHR, SRA

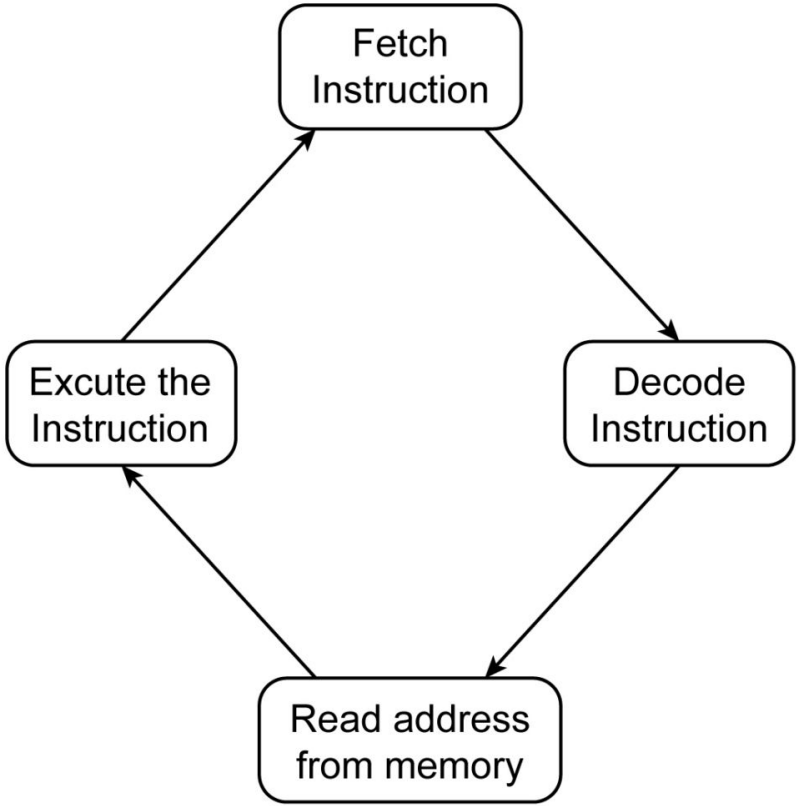
EPIC
Institute of Technology
Powered by epam

Instructions

1. Arithmetic instructions
2. Logical instructions
3. Data transfer instructions
4. Branch instructions
5. String manipulation instructions
6. System instructions

EPIC
Institute of Technology
Powered by epam

Instruction Cycle



EPIC
Institute of Technology
Powered by epam

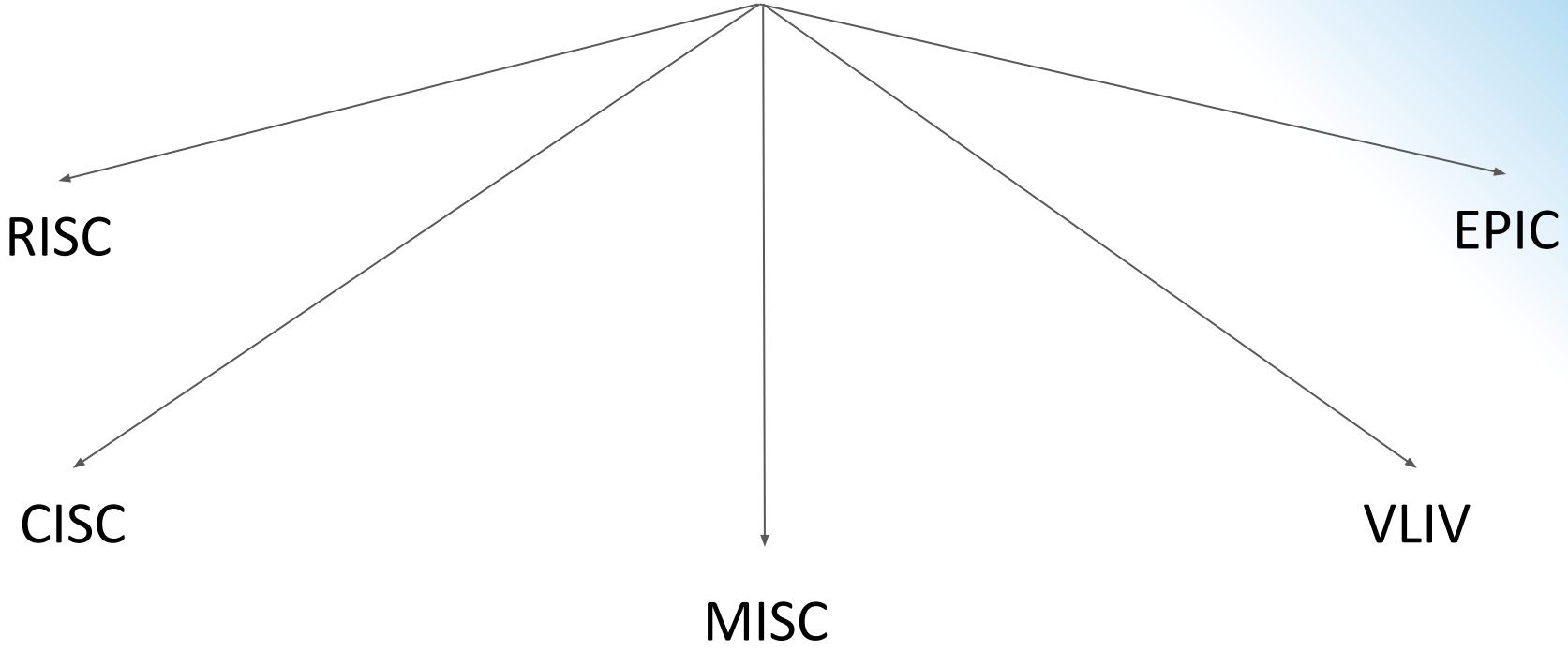
02

Architecture

A little topic about different architectures

EPIC
Institute of Technology
Powered by epam

Architecture



EPIC Institute of Technology
Powered by <epam>

RISC vs CISC

1. RISC stands for Reduced Instruction Set Computer.	1. CISC stands for Complex Instruction Set Computer.
2. RISC processors have simple instructions taking about one clock cycle. The average clock cycle per instruction (CPI) is 1.5	2. CSIC processor has complex instructions that take up multiple clocks for execution. The average clock cycle per instruction (CPI) is in the range of 2 and 15.
3. Performance is optimized with more focus on software	3. Performance is optimized with more focus on hardware.
4. It has no memory unit and uses a separate hardware to implement instructions..	4. It has a memory unit to implement complex instructions.
5. It has a hard-wired unit of programming.	5. It has a microprogramming unit.
6. The instruction set is reduced i.e. it has only a few instructions in the instruction set. Many of these instructions are very primitive.	6. The instruction set has a variety of different instructions that can be used for complex operations.
7. The instruction set has a variety of different instructions that can be used for complex operations.	7. CISC has many different addressing modes and can thus be used to represent higher-level programming language statements more efficiently.

CISC

PROS

1. High performance
2. Ease of programming in asm
3. Lower memory usage

CONS

1. Architecture complexity
2. Larger processor size
3. Low clock frequencies
4. Low IPC

EPIC
Institute of Technology
Powered by <epam>

EPIC
Institute of Technology
Powered by epam

RISC

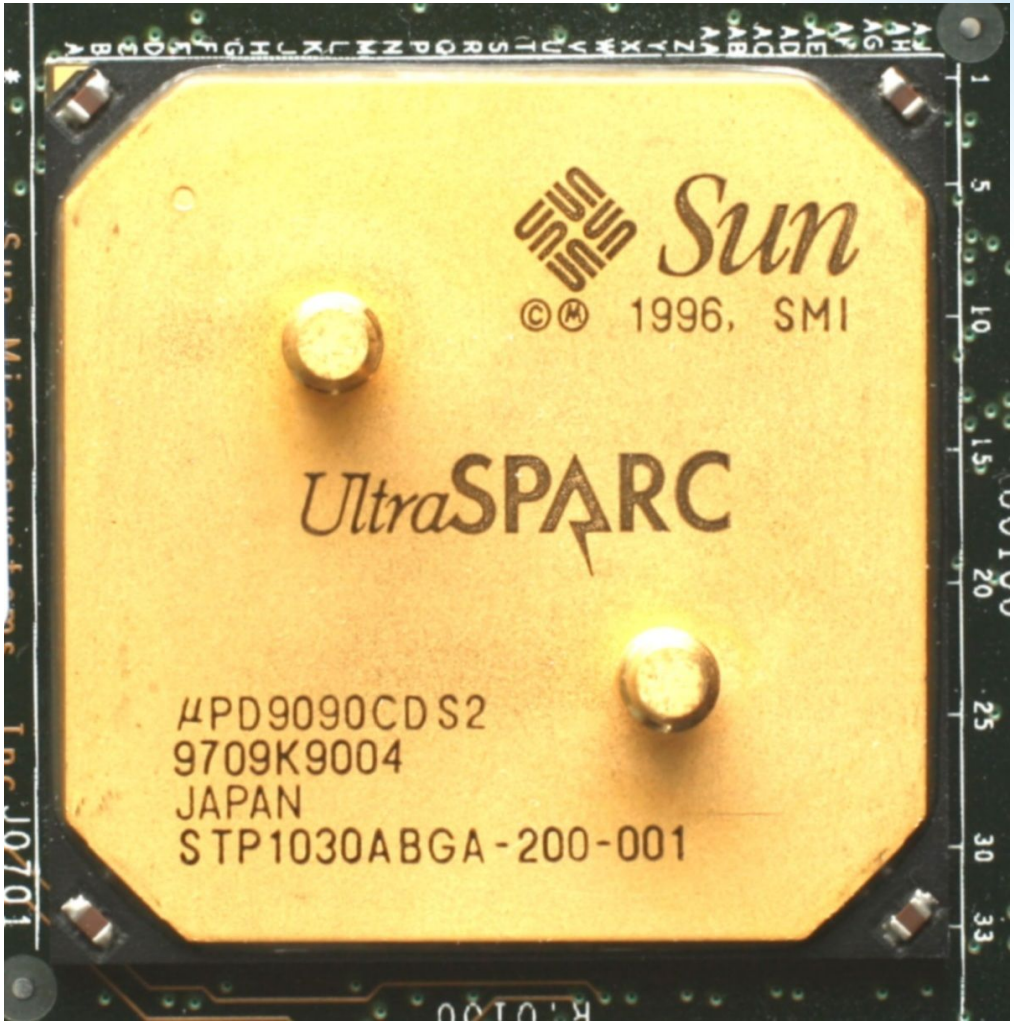
PROS

1. Easy to create architecture
2. Smaller processor size
3. High IPC
4. Cache optimizations because of Memory
5. Easy to create compiler

CONS

1. Less instruction
2. Hard to write in asm
3. Higher memory usage

EPIC
Institute of Technology
Powered by <epam>



EPIC
Institute of Technology
Powered by epam

MISC

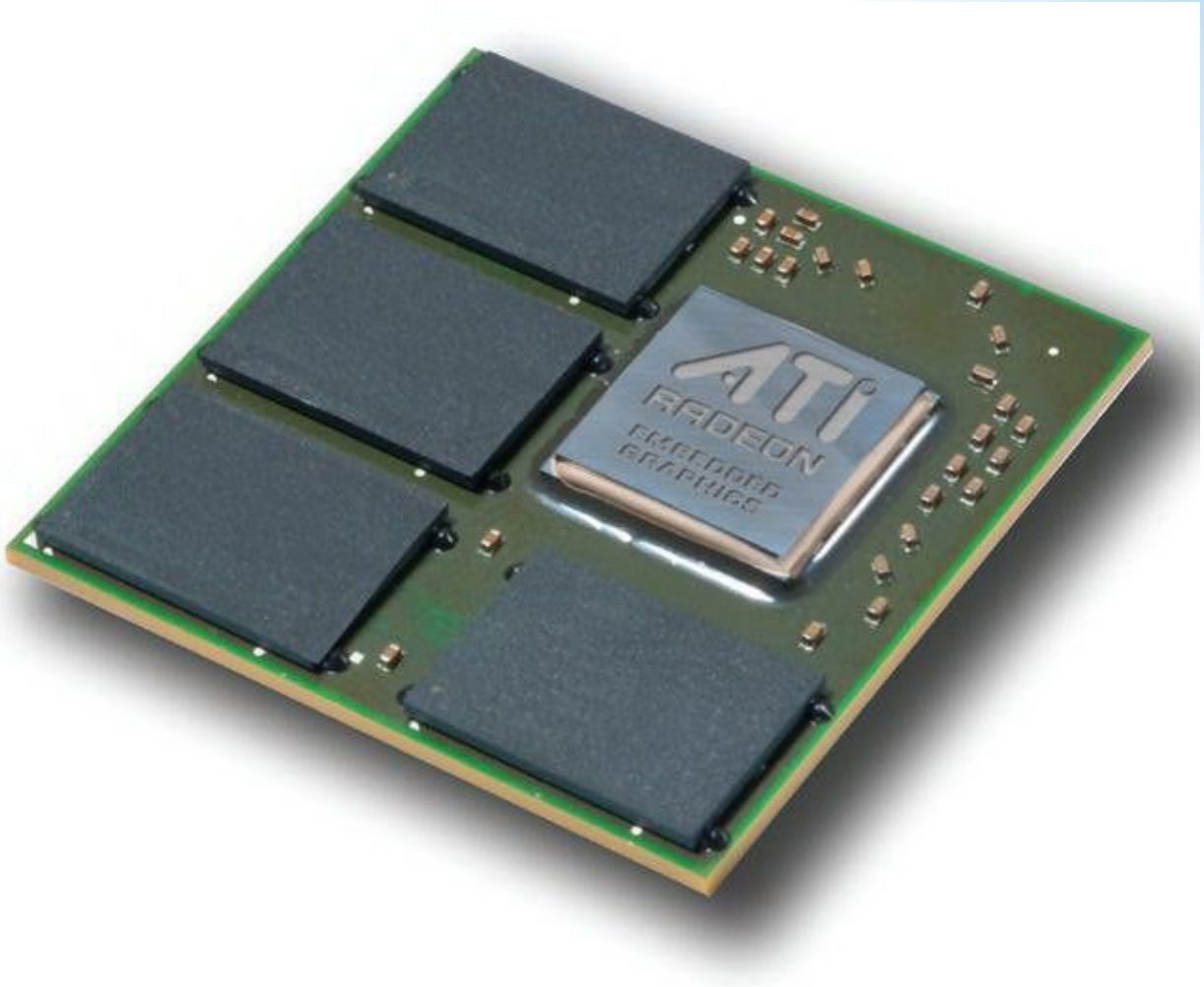
PROS

1. Simplicity of architecture
2. Low cost
3. Low power consumption

CONS

1. Limited functionality
2. Complexity of programming in asm
3. Limited support

EPIC
Institute of Tech
Powered by <epam>



EPIC
Institute of Technology
Powered by epam

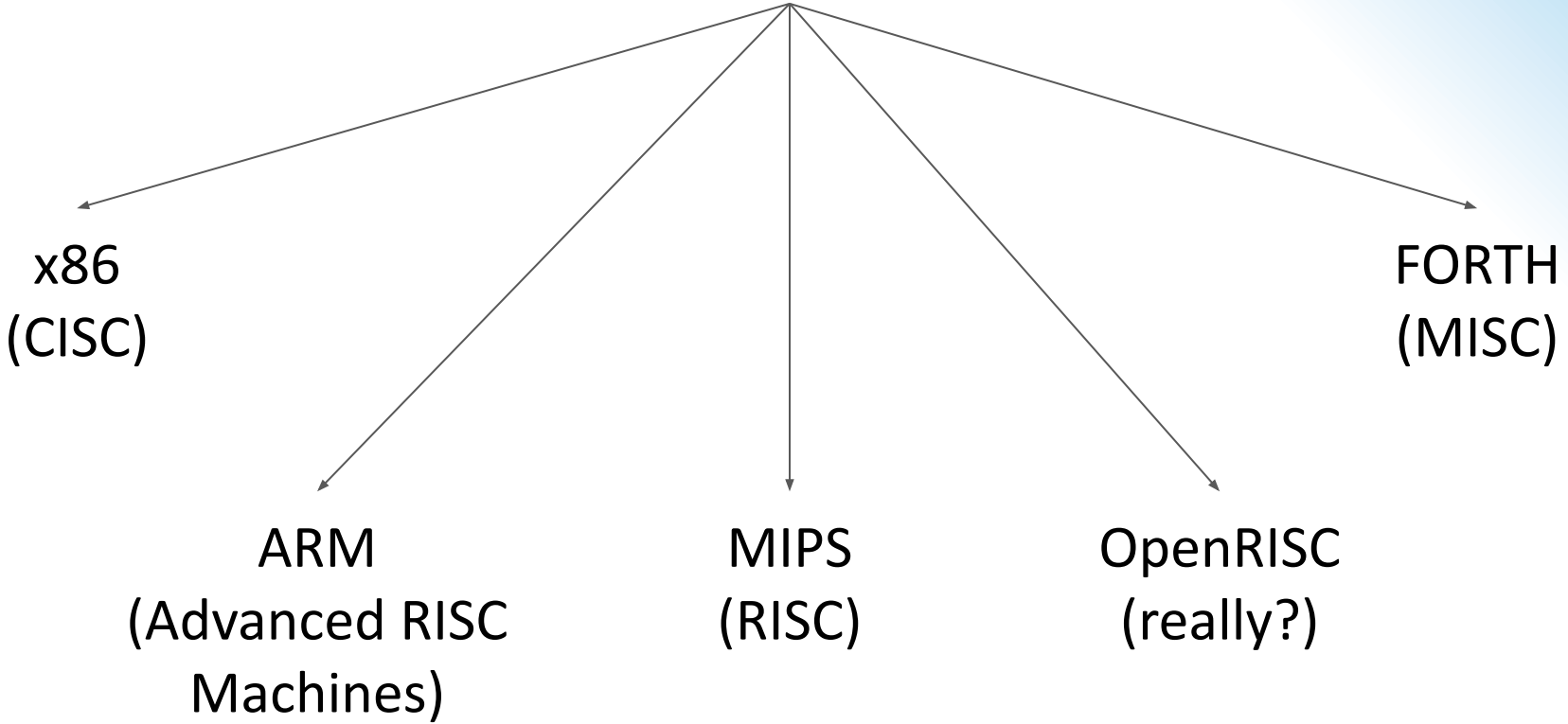
03

ISA

A little topic about different ISA

EPIC
Institute of Technology
Powered by epam

Architecture



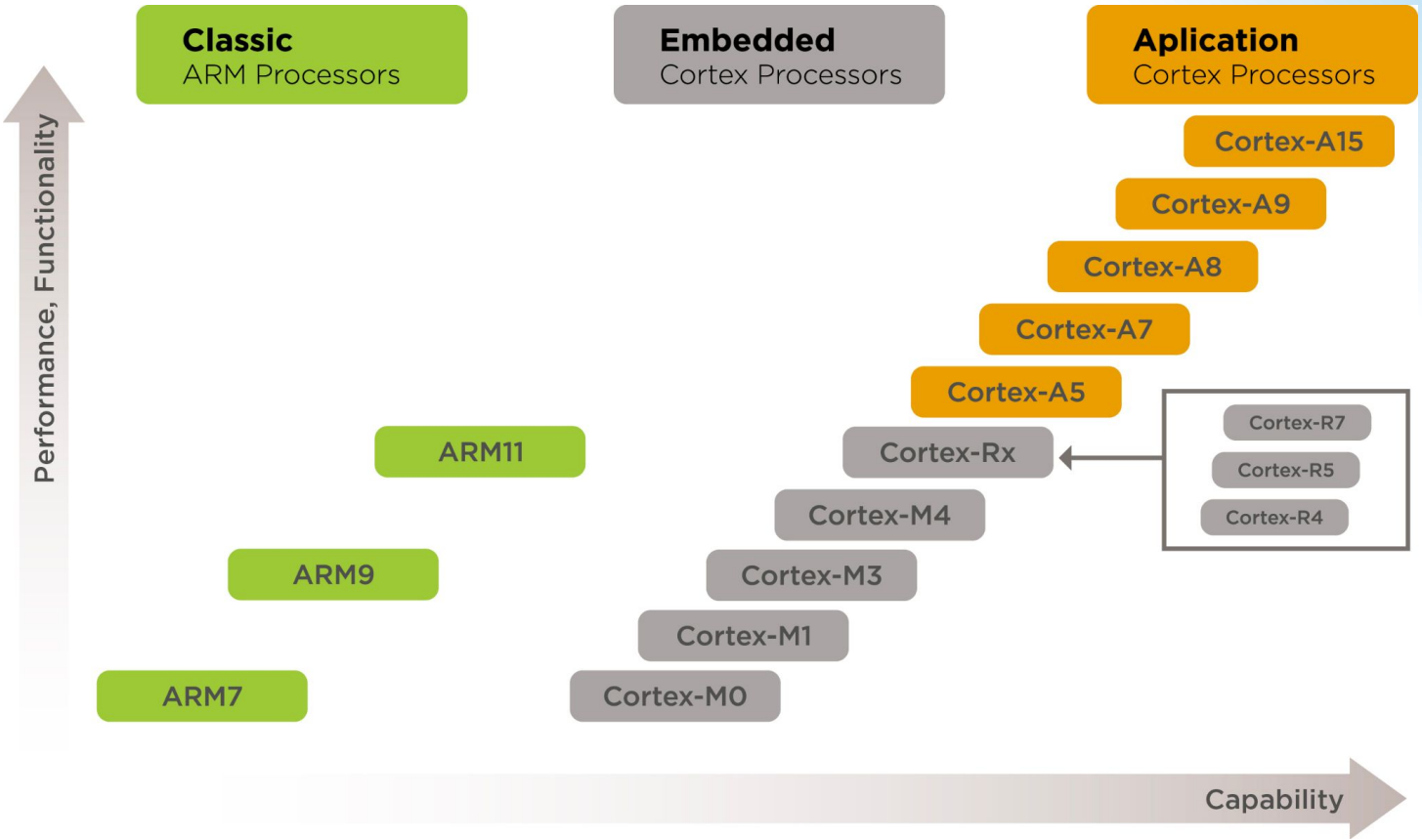
EPIC
Institute of Technology
Powered by <epam>

x86 PROCESSORS (from Intel)

Bits	Family	Clock Speed (approx.)	Bus Size (bits)	Max RAM	Storage Range	OS
64	Xeon	4.3GHz	64	3TB	500GB-10TB	Windows: 10, 8, 7 XP, 2000 NT, 98 95, 3.x Linux Mac OS X SCO Unix Solaris
	Core i9	3.3GHz		128GB		
	Core i3, i5, i7	3.3GHz		64GB		
	Core 2 Duo	2.6GHz				
	Pentium 4	3.8GHz				
	Pentium D	3.4GHz				
32	Core Duo	2.2GHz				
	Pentium 4	2.8GHz				
	Xeon	3.2GHz				
	Celeron	2.4GHz		4GB		
	Pentium III	1.2GHz				
	Pentium II	450MHz				
	Pentium Pro	233MHz		64GB	500MB-60GB	DOS
	Pentium	200MHz		4GB		DR DOS
	486DX	100MHz	32		200 - 500MB	OS/2
	486SX	40MHz				Misc DOS
	386DX	40MHz			60 - 200MB	Multiuser
	386SX	33MHz				
	386SL	25MHz				
16	286	12MHz	16	16MB	20-80MB	DOS DR DOS Win 3.x OS/2 1.x
	8086	10MHz	8	1MB	10-20MB	DOS DR DOS
	8088	5MHz				

EPIC
Institute of Technology
Powered by <epam>

ARM examples



x86

1. MOV destination, source
2. JMP label
3. No
4. CMPXCHG

ARM

1. MOV{condition, change_flags} source, destination
2. B label
3. MLS R1, R2, R3, R4
4. No

ARM

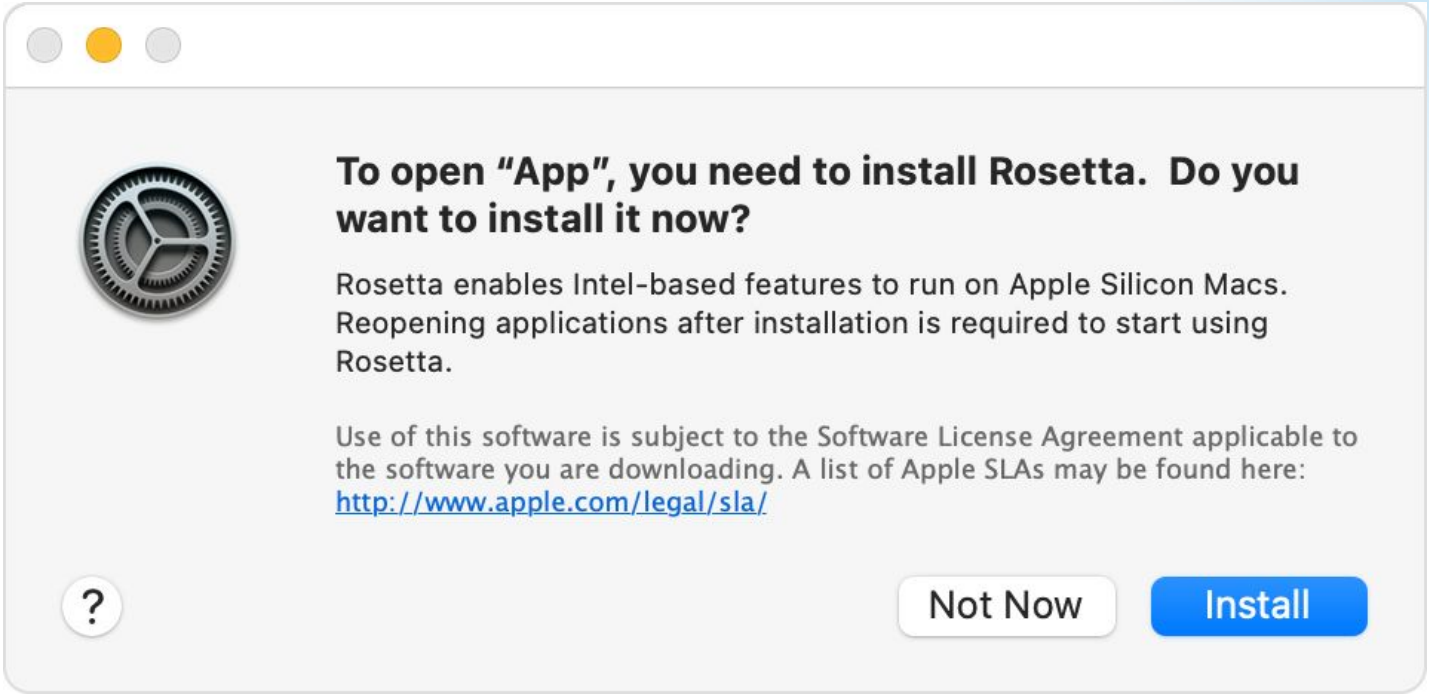
1. More support

OpenRISC

1. Open source
2. Easy to adapt

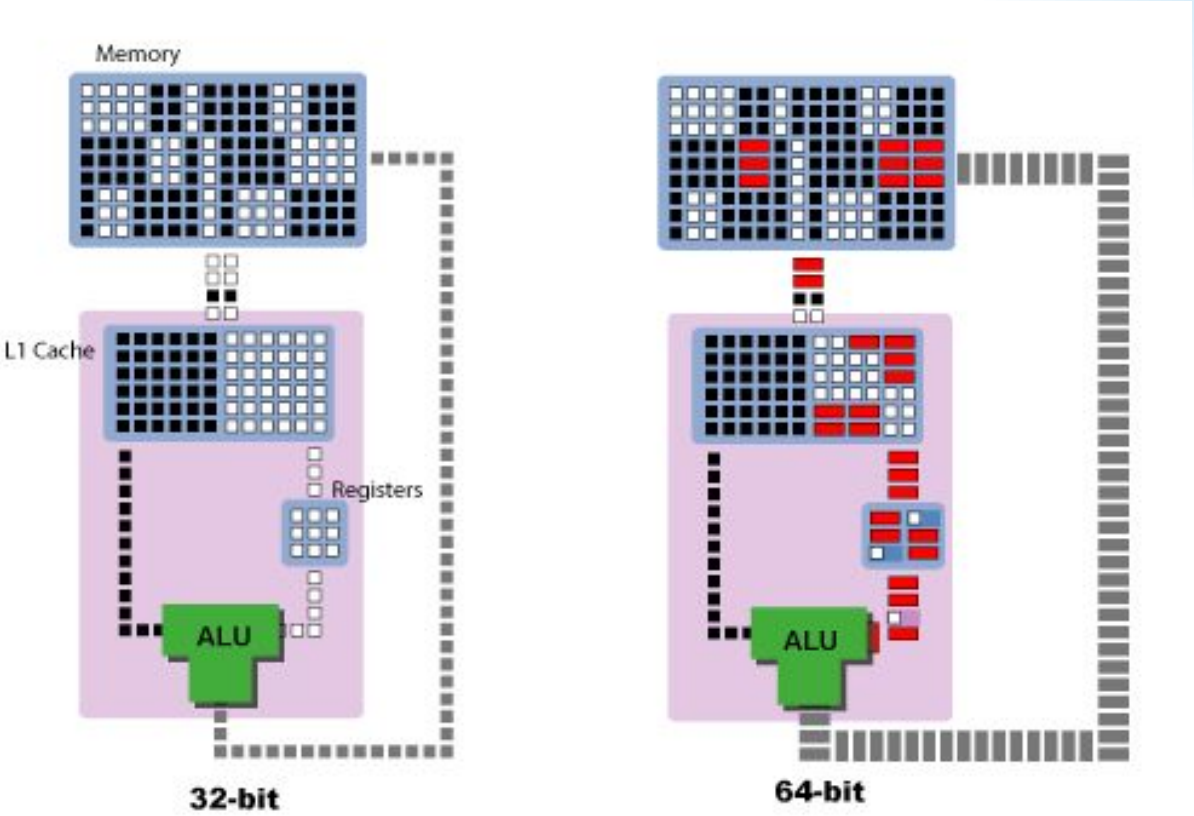
EPIC
Institute of Technology
Powered by <epam>

Rosetta



EPIC
Institute of Technology
Powered by epam

32 vs 64 bits



EPIC
Institute of Technology
Powered by epam

04

ASM

A little topic about theory of ASM

EPIC
Institute of Technology
Powered by epam

ASM

1. NASM (Netwide ASM)
2. MASM (Microsoft Macro ASM)
3. GAS (GNU Assembler)

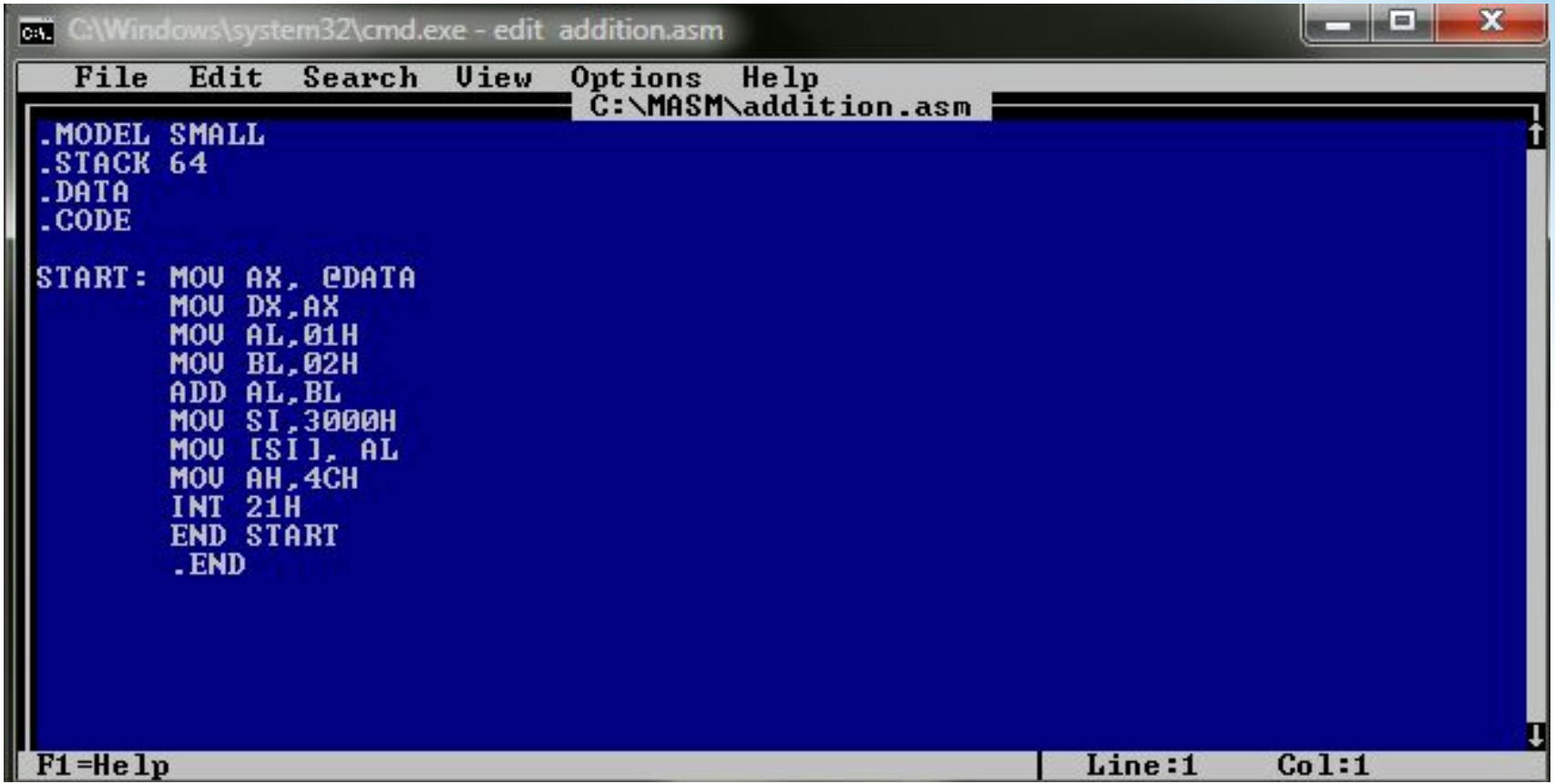
EPIC
Institute of Technology
Powered by <epam>

NASM

```
.ebios_check_done:
%else
    inc al
    mov [ebios_present], al
%endif

    ; Since this code may not always reside in the MBR, always start by
    ; loading the MBR to kMBRBuffer.
    mov al, 1                   ; load one sector
    xor bx, bx
    mov es, bx                  ; MBR load segment = 0
    mov bx, MBRBufferAddr       ; MBR load address
    mov si, bx                  ; pointer to fake partition entry
%if CHS_SUPPORT
    mov word [si], 0000h        ; CHS DX: head = 0
    mov word [si+2], 0001h      ; CHS CX: cylinder = 0, sector = 1
%endif
    mov dword [si+partition.lba], 00000000h ; LBA sector 0
    call load_sectors
    jc .next_drive              ; MBR load error

    ; Look for the booter partition in the MBR partition table,
    ; which is at offset kMBRPartTable.
    mov di, MBRPartitionTableAddr   ; pointer to partition table
    call find_boot                  ; will not return on success

.next_drive:
    inc dl                      ; next drive number
    test dl, 4                  ; went through all 4 drives?
    jz .loop                    ; not yet, loop again

hang: hlt
    jmp short hang
```

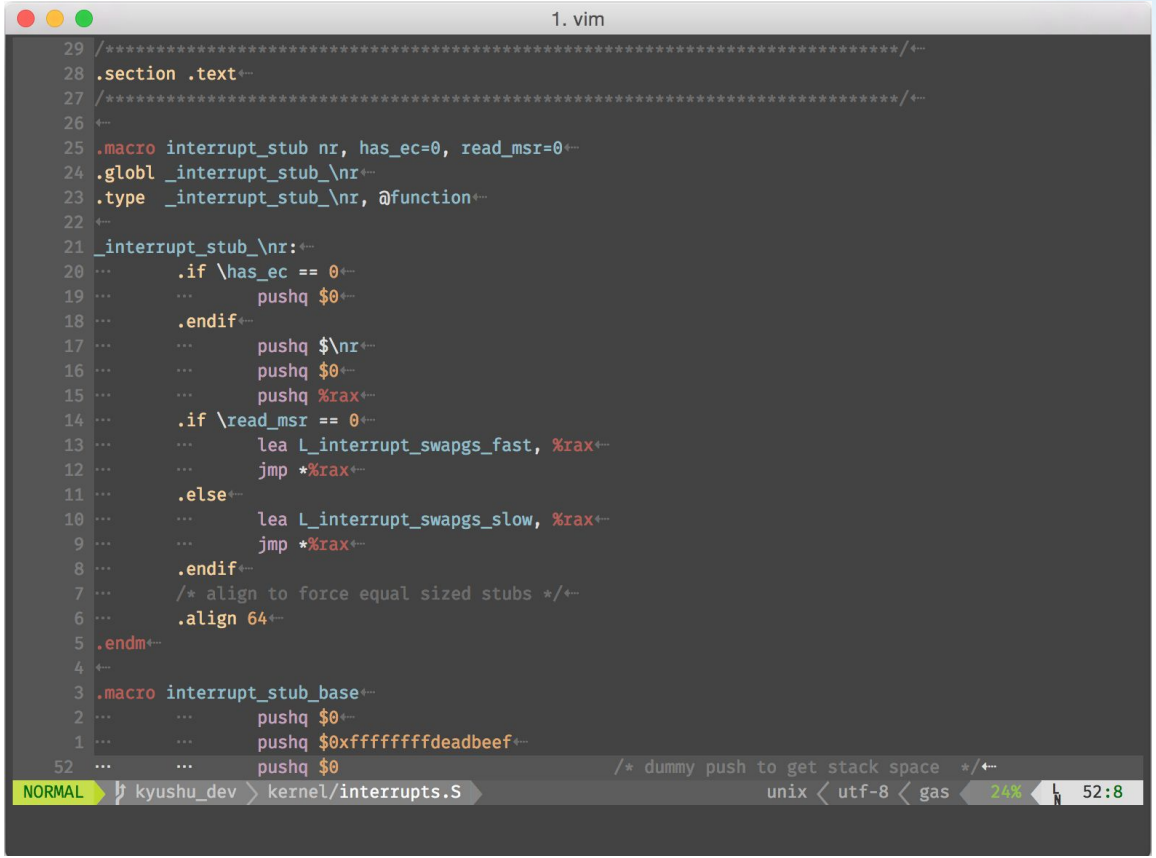
EPIC
Institute of Technology
Powered by <epam>

MASM



EPIC
Institute of Technology
Powered by <epam>

GAS



GAS	NASM
1. .align	1. times
2. .string	2. %define
3. .type, .size	3. %include

EPIC
Institute of Technology
Powered by epam

How?

0. Linux

1. VirtualBox
2. Server
3. Docker

EPIC Institute of Technology
Powered by epam

ASM code

1. Directives - "."
2. Global symbols - ".global" or ".globl".
3. Sections - ".section" or simply the name of the section, for example ".text".
4. Labels
5. Instructions
6. Comments - Comments in assembly language are denoted by the "#" symbol.

EPIC Institute of Technology
Powered by <epam>

GAS -> AT&T

Hello world

```
.section .data      # Define the data section where the data will be stored
hello:              # Define a label called "hello"
    .ascii "Hello, world!\n"    # Define a null-terminated string of characters

.section .text      # Define the text section where the executable code will be stored
.globl _start       # Make the _start symbol globally accessible
_start:             # Define the _start label for the entry point of the program
    movl $4, %eax   # Move the system call number for "write" into the eax register
    movl $1, %ebx   # Move the file descriptor for stdout into the ebx register
    movl $hello, %ecx   # Move the address of the string to be printed into the ecx register
    movl $15, %edx  # Move the length of the string to be printed into the edx register
    int $0x80       # Invoke the kernel interrupt to perform the system call

    movl $1, %eax   # Move the system call number for "exit" into the eax register
    xorl %ebx, %ebx # Set the exit status to 0
    int $0x80       # Invoke the kernel interrupt to perform the system call
```

EPIC Institute of Technology
Powered by epam

INTEL

Hello world

HelloWorld.asm 429w94eyh

```
1  section .data
2  hello:
3    db 'Hello, world!', 0Ah
4
5  section .text
6  global _start
7  _start:
8    mov eax, 4            ; Move the system call number for "write" into the eax register
9    mov ebx, 1            ; Move the file descriptor for stdout into the ebx register
10   mov ecx, hello        ; Move the address of the string to be printed into the ecx register
11   mov edx, 13           ; Move the length of the string to be printed into the edx register
12   int 0x80              ; Invoke the kernel interrupt to perform the system call
13
14   mov eax, 1            ; Move the system call number for "exit" into the eax register
15   xor ebx, ebx          ; Set the exit status to 0
16   int 0x80              ; Invoke the kernel interrupt to perform the system call
17
```

https://onecompiler.com/assembly/429w94eyh

EPIC
Institute of Technology
Powered by epam

Registers

1. General purpose registers: %eax, %ebx, %ecx, %edx, %esi, %edi, %ebp, %esp.
2. Segment registers: %cs, %ds, %es, %ss, %fs, %gs - PARTIALLY LEGACY
3. Status register: %eflags.
4. FPU registers: %st(0) - %st(7).
5. MMX registers: %mm0 - %mm7. - LEGACY
6. SSE registers: %xmm0 - %xmm7.

EPIC
Institute of Technology
Powered by epam

Hello world

```
.section .data      # Define the data section where the data will be stored
hello:              # Define a label called "hello"
    .ascii "Hello, world!\n"    # Define a null-terminated string of characters

.section .text      # Define the text section where the executable code will be stored
.globl _start       # Make the _start symbol globally accessible
_start:             # Define the _start label for the entry point of the program
    movl $4, %eax   # Move the system call number for "write" into the eax register
    movl $1, %ebx   # Move the file descriptor for stdout into the ebx register
    movl $hello, %ecx   # Move the address of the string to be printed into the ecx register
    movl $15, %edx  # Move the length of the string to be printed into the edx register
    int $0x80       # Invoke the kernel interrupt to perform the system call

    movl $1, %eax   # Move the system call number for "exit" into the eax register
    xorl %ebx, %ebx # Set the exit status to 0
    int $0x80       # Invoke the kernel interrupt to perform the system call
```

EPIC Institute of Technology
Powered by epam

Operand

1. Register - for example %eax, %ebx, %ecx
2. Memory

   a) Absolute - 0x1000, (%eax)

   b) Relative - label(%eax), 4(%ebp), 8(%eax, %ecx, 4)

1. Immediate operand - $0x100, $'a'
2. Flags register - %eflags

EPIC
Institute of Technology
Powered by epam

Hello world

```
.section .data      # Define the data section where the data will be stored
hello:              # Define a label called "hello"
    .ascii "Hello, world!\n"    # Define a null-terminated string of characters

.section .text      # Define the text section where the executable code will be stored
.globl _start       # Make the _start symbol globally accessible
_start:             # Define the _start label for the entry point of the program
    movl $4, %eax   # Move the system call number for "write" into the eax register
    movl $1, %ebx   # Move the file descriptor for stdout into the ebx register
    movl $hello, %ecx   # Move the address of the string to be printed into the ecx register
    movl $15, %edx  # Move the length of the string to be printed into the edx register
    int $0x80       # Invoke the kernel interrupt to perform the system call

    movl $1, %eax   # Move the system call number for "exit" into the eax register
    xorl %ebx, %ebx # Set the exit status to 0
    int $0x80       # Invoke the kernel interrupt to perform the system call
```

EPIC Institute of Technology
Powered by epam

Arithmetic instructions

1. add
2. sub
3. mul
4. div

EPIC
Institute of Technology
Powered by epam

Hello world

```
.section .data      # Define the data section where the data will be stored
hello:              # Define a label called "hello"
    .ascii "Hello, world!\n"    # Define a null-terminated string of characters

.section .text      # Define the text section where the executable code will be stored
.globl _start       # Make the _start symbol globally accessible
_start:             # Define the _start label for the entry point of the program
    movl $4, %eax   # Move the system call number for "write" into the eax register
    movl $1, %ebx   # Move the file descriptor for stdout into the ebx register
    movl $hello, %ecx   # Move the address of the string to be printed into the ecx register
    movl $15, %edx  # Move the length of the string to be printed into the edx register
    int $0x80       # Invoke the kernel interrupt to perform the system call

    movl $1, %eax   # Move the system call number for "exit" into the eax register
    xorl %ebx, %ebx # Set the exit status to 0
    int $0x80       # Invoke the kernel interrupt to perform the system call
```

EPIC
Institute of Technology
Powered by epam

Logic instructions

1. and
2. or
3. xor

EPIC Institute of Technology
Powered by epam

Data movement instructions

1. mov
2. push
3. pop
4. lea
5. xchg
6. movsb

EPIC
Institute of Technology
Powered by epam

Hello world

```
.section .data      # Define the data section where the data will be stored
hello:              # Define a label called "hello"
    .ascii "Hello, world!\n"    # Define a null-terminated string of characters

.section .text      # Define the text section where the executable code will be stored
.globl _start       # Make the _start symbol globally accessible
_start:             # Define the _start label for the entry point of the program
    movl $4, %eax   # Move the system call number for "write" into the eax register
    movl $1, %ebx   # Move the file descriptor for stdout into the ebx register
    movl $hello, %ecx   # Move the address of the string to be printed into the ecx register
    movl $15, %edx  # Move the length of the string to be printed into the edx register
    int $0x80       # Invoke the kernel interrupt to perform the system call

    movl $1, %eax   # Move the system call number for "exit" into the eax register
    xorl %ebx, %ebx # Set the exit status to 0
    int $0x80       # Invoke the kernel interrupt to perform the system call
```

EPIC
Institute of Technology
Powered by epam

Control instructions

1. conditional jump

   (je, jne, jl, jle, jg, jge)
2. loop
3. call
4. jmp

EPIC
Institute of Technology
Powered by epam

Labels example

```
start:
    mov eax, 1
    mov ebx, 2
    add eax, ebx
    jmp end_prog
exit:
    mov eax, 0
end_prog:
    cmp eax, 0
    je exit
```

EPIC
Institute of Technology
Powered by epam

05

ASM examples

A little topic about ASM examples

EPIC
Institute of Technology
Powered by epam

Simple examples

```
 1  addl    SRC, DST        /* DST += SRC */
 2  subl    SRC, DST        /* DST -= SRC */
 3  incl    DST             /* ++DST */
 4  decl    DST             /* --DST */
 5  negl    DST             /* DST = -DST */
 6  movl    SRC, DST        /* DST = SRC */
 7  imull   SRC             /* (%eax,%edx) = %eax * SRC - singed */
 8  mull    SRC             /* (%eax,%edx) = %eax * SRC - unsigned */
 9  andl    SRC, DST        /* DST &= SRC */
10  orl     SRC, DST        /* DST |= SRC */
11  xorl    SRC, DST        /* DST ^= SRC */
12  notl    DST             /* DST = ~DST */
```

EPIC
Institute of Technology
Powered by <epam>

32-bits

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)
0	restart_syscall	man/ cs/	0x00	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-
2	fork	man/ cs/	0x02	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode
6	close	man/ cs/	0x06	unsigned int fd	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp
12	chdir	man/ cs/	0x0c	const char *filename	-	-
13	time	man/ cs/	0x0d	time_t *tloc	-	-
14	mknod	man/ cs/	0x0e	const char *filename	umode_t mode	unsigned dev
15	chmod	man/ cs/	0x0f	const char *filename	umode_t mode	-
16	lchown	man/ cs/	0x10	const char *filename	uid_t user	gid_t group

128	init_module	man/ cs/	0x80	void *umod	unsigned long len	const char *uargs

EPIC
Institute of Technology
Powered by epam

Hello world

```
.section .data      # Define the data section where the data will be stored
hello:              # Define a label called "hello"
    .ascii "Hello, world!\n"    # Define a null-terminated string of characters

.section .text      # Define the text section where the executable code will be stored
.globl _start       # Make the _start symbol globally accessible
_start:             # Define the _start label for the entry point of the program
    movl $4, %eax   # Move the system call number for "write" into the eax register
    movl $1, %ebx   # Move the file descriptor for stdout into the ebx register
    movl $hello, %ecx   # Move the address of the string to be printed into the ecx register
    movl $15, %edx  # Move the length of the string to be printed into the edx register
    int $0x80       # Invoke the kernel interrupt to perform the system call

    movl $1, %eax   # Move the system call number for "exit" into the eax register
    xorl %ebx, %ebx # Set the exit status to 0
    int $0x80       # Invoke the kernel interrupt to perform the system call
```

EPIC Institute of Technology
Powered by epam

64-bits

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode
3	close	man/ cs/	0x03	unsigned int fd	-	-
4	stat	man/ cs/	0x04	const char *filename	struct __old_kernel_stat *statbuf	-
5	fstat	man/ cs/	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-
6	lstat	man/ cs/	0x06	const char *filename	struct __old_kernel_stat *statbuf	-
7	poll	man/ cs/	0x07	struct pollfd *ufds	unsigned int nfds	int timeout
8	lseek	man/ cs/	0x08	unsigned int fd	off_t offset	unsigned int whence
9	mmap	man/ cs/	0x09	?	?	?
10	mprotect	man/ cs/	0x0a	unsigned long start	size_t len	unsigned long prot
11	munmap	man/ cs/	0x0b	unsigned long addr	size_t len	-
12	brk	man/ cs/	0x0c	unsigned long brk	-	-
13	rt_sigaction	man/ cs/	0x0d	int	const struct sigaction *	struct sigaction *
14	rt_sigprocmask	man/ cs/	0x0e	int how	sigset_t *set	sigset_t *oset
15	rt_sigreturn	man/ cs/	0x0f	?	?	?

60	exit	man/ cs/	0x3c	int error_code	-	-

EPIC
Institute of Technology
Powered by epam

Hello world

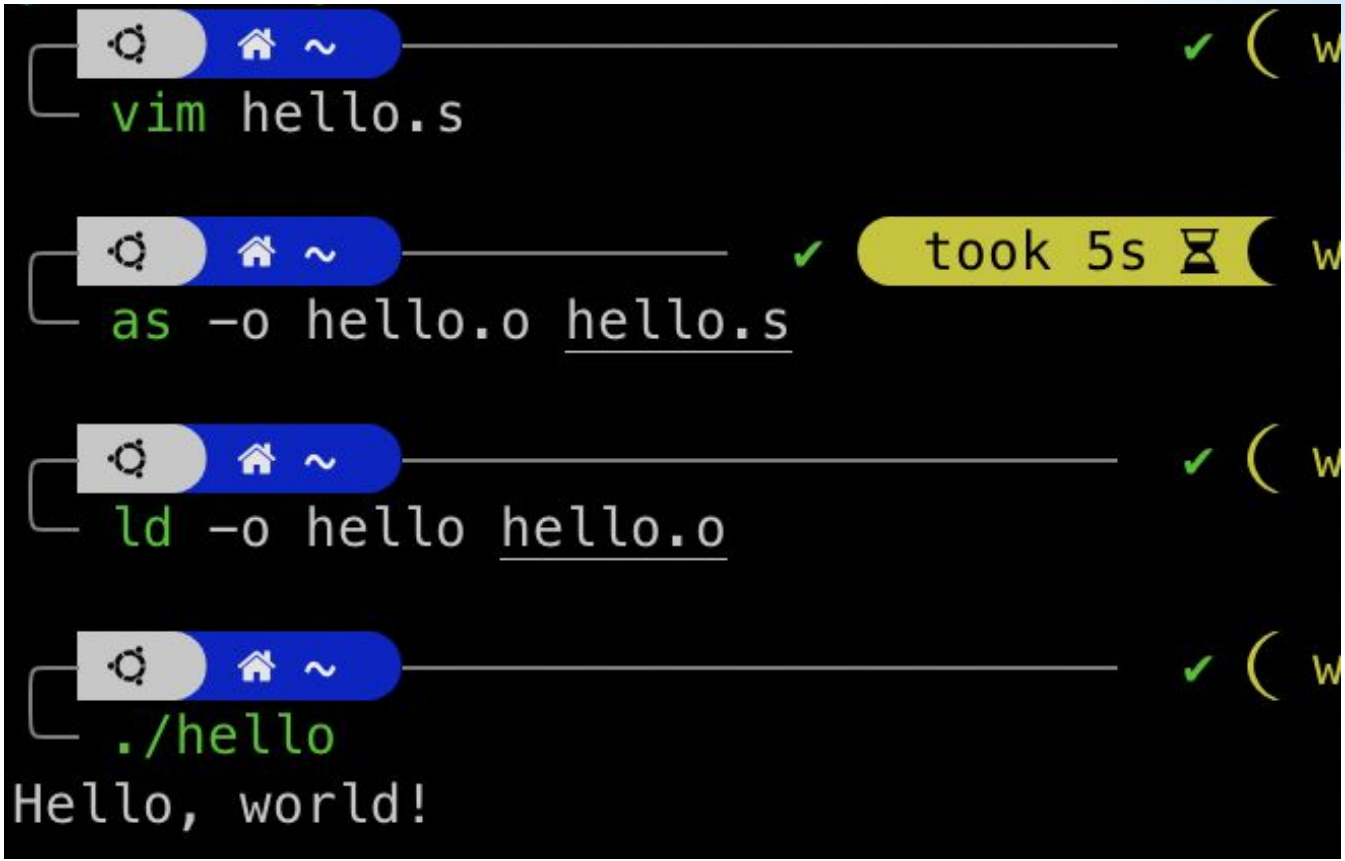
```
.section .data
hello:
    .ascii "Hello, world!\n"

.section .text
.globl _start
_start:
    mov $1, %rax
    mov $1, %rdi
    mov $hello, %rsi
    mov $15, %rdx
    syscall

    mov $60, %rax
    xor %rdi, %rdi
    syscall
```

EPIC
Institute of Technology
Powered by <epam>

Machine code

EPIC
Institute of Technology
Powered by <epam>

Run



-fpic

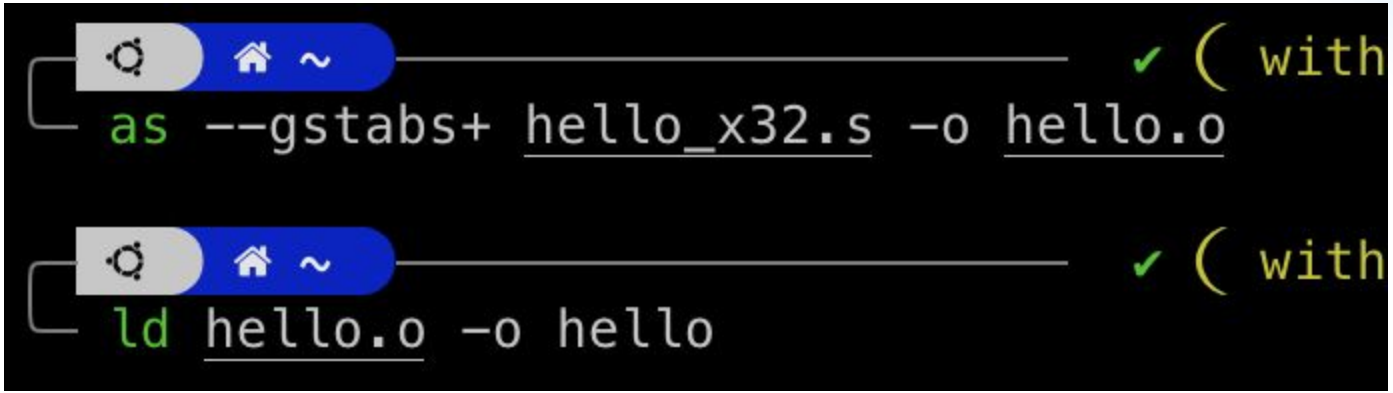
Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that **-fpic** does not work; in that case, recompile with **-fPIC** instead. (These maximums are 8k on the SPARC, 28k on AArch64 and 32k on the m68k and RS/6000. The x86 has no such limit.)

Position-independent code requires special support, and therefore works only on certain machines. For the x86, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent.

When this flag is set, the macros `"__pic__"` and `"__PIC__"` are defined to 1.

-no-pie

Don't produce a position independent executable.


~
✔ (with
as --gstabs+ hello_x32.s -o hello.o
~
✔ (with
ld hello.o -o hello

EPIC
Institute of Technology
Powered by epam

06

Programming languages

A little topic about “why my code is working”

EPIC
Institute of Technology
Powered by epam

ASM -> Binary

Machine code	Assembly code	Description
001 1 000010	LOAD #2	Load the value 2 into the Accumulator
010 0 001101	STORE 13	Store the value of the Accumulator in memory location 13
001 1 000101	LOAD #5	Load the value 5 into the Accumulator
010 0 001110	STORE 14	Store the value of the Accumulator in memory location 14
001 0 001101	LOAD 13	Load the value of memory location 13 into the Accumulator
011 0 001110	ADD 14	Add the value of memory location 14 to the Accumulator
010 0 001111	STORE 15	Store the value of the Accumulator in memory location 15
111 0 000000	HALT	Stop execution

EPIC Institute of Technology
Powered by epam

ASM -> Binary

Machine Code	Instruction Calls	Assembly Code
fc01 0113	addi(X(2),X(2),-64)	addi sp, sp, -
0281 3c23	sd(X(8),X(2),56)	sd s0, 56(sp
0401 0413	addi(X(8),X(2),64)	addi s0, sp, 6
fca4 3c23	sd(X(10),X(8),-40)	sd a0, -40(s
0005 8793	addi(X(15),X(11),0)	addi a5, a1, 0
fcc4 3423	sd(X(12),X(8),-56)	sd a2, -56(s
fcf4 2a23	sw(X(15),X(8),-44)	sw a5, -44(s
fd84 3783	ld(X(15),X(8),-40)	ld a5, -40(s
fef4 3423	sd(X(15),X(8),-24)	sd a5, -24(s
fc84 3783	ld(X(15),X(8),-56)	ld a5, -56(s
fef4 3023	sd(X(15),X(8),-32)	sd a5, -32(s
01c0 006f	jal(X(0),14))	jal zero, 14
fe84 3783	ld(X(15),X(8),-24)	ld a5, -24(s
0017 8713	addi(X(14),X(15),1)	addi a4, a5, 1
fee4 3423	sd(X(14),X(8),-24)	sd a4, -24(s
fd44 2703	lw(X(14),X(8),-44)	lw a4, -44(s
0ff7 7713	andi(X(14),X(14),255)	andi a4, a4, 2
00e7 8023	sb(X(14),X(15),0)	sb a4, 0(a5)
fc84 3783	ld(X(15),X(8),-56)	ld a5, -56(s
fff7 8713	addi(X(14),X(15),-1)	addi a4, a5, -
fce4 3423	sd(X(14),X(8),-56)	sd a4, -56(s
fc07 9ee3	bne(X(15),X(0),-18)	bne a5, zero,
fd84 3783	ld(X(15),X(8),-40)	ld a5, -40(s
0007 8513	addi(X(10),X(15),0)	addi a0, a5, 0
0381 3403	ld(X(8),X(2),56)	ld s0, 56(sp
0401 0113	addi(X(2),X(2),64)	addi sp, sp, 6
0000 8067	jalr(X(0),X(1),0)	jalr zero, ra,

EPIC
Institute of Technology
Powered by epam

Hello world

```
.section .data
hello:
    .ascii "Hello, world!\n"

.section .text
.globl _start
_start:
    mov $1, %rax
    mov $1, %rdi
    mov $hello, %rsi
    mov $15, %rdx
    syscall

    mov $60, %rax
    xor %rdi, %rdi
    syscall
```

EPIC
Institute of Technology
Powered by <epam>

Disassembling

```
~                                                    1 ✘  with glebodin@glebodinvir
objdump -d hello_x64

hello_x64:     file format elf64-x86-64


Disassembly of section .text:

000000000040010c <_start>:
  40010c:       48 c7 c0 01 00 00 00    mov    $0x1,%rax
  400113:       48 c7 c7 01 00 00 00    mov    $0x1,%rdi
  40011a:       48 c7 c6 36 01 60 00    mov    $0x600136,%rsi
  400121:       48 c7 c2 0f 00 00 00    mov    $0xf,%rdx
  400128:       0f 05                   syscall
  40012a:       48 c7 c0 3c 00 00 00    mov    $0x3c,%rax
  400131:       48 31 ff                xor    %rdi,%rdi
  400134:       0f 05                   syscall
```

EPIC
Institute of Technology
Powered by epam

C++

```
#include <iostream>

using namespace std;

int main() {
        int x = 10, y = 15;
        int sum = x + y;
        cout << sum;
}
~
```

EPIC
Institute of Technology
Powered by epam

Disassembling

```
00000000000007ca <main>:
 7ca:   55                      push   %rbp
 7cb:   48 89 e5                mov    %rsp,%rbp
 7ce:   48 83 ec 10             sub    $0x10,%rsp
 7d2:   c7 45 f4 0a 00 00 00    movl   $0xa,-0xc(%rbp)
 7d9:   c7 45 f8 0f 00 00 00    movl   $0xf,-0x8(%rbp)
 7e0:   8b 55 f4                mov    -0xc(%rbp),%edx
 7e3:   8b 45 f8                mov    -0x8(%rbp),%eax
 7e6:   01 d0                   add    %edx,%eax
 7e8:   89 45 fc                mov    %eax,-0x4(%rbp)
 7eb:   8b 45 fc                mov    -0x4(%rbp),%eax
 7ee:   89 c6                   mov    %eax,%esi
 7f0:   48 8d 3d 29 08 20 00    lea    0x200829(%rip),%rdi        # 201020 <_ZSt4cout@@G
LIBCXX_3.4>
 7f7:   e8 a4 fe ff ff          callq  6a0 <_ZNSolsEi@plt>
 7fc:   b8 00 00 00 00          mov    $0x0,%eax
 801:   c9                      leaveq
 802:   c3                      retq
```

EPIC Institute of Technology
Powered by epam

Inline assembler

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 15, y = 10, sum = 0;
    cout << "x = " << x << ", y = " << y << endl;

    __asm__(
        "addl %%ebx, %%eax"   //eax = eax + ebx (eax = x + y)
        :"=a"(sum)            //output: we take sum from eax
        :"a"(x), "b"(y)       //input: we put x into eax, z into ebx
    );

    cout << "y + z = " << sum << endl;
    return 0;
}
```

Успешно #stdin #stdout 0s 5444KB

stdin

```
Standard input is empty
```

stdout

```
x = 15, y = 10
y + z = 25
```

EPIC
Institute of Technology
Powered by epam

```
glebodin@glebodinvirtual: ~ (-zsh)
000000000000090a <main>:
 90a:   55                      push   %rbp
 90b:   48 89 e5                mov    %rsp,%rbp
 90e:   53                      push   %rbx
 90f:   48 83 ec 18             sub    $0x18,%rsp
 913:   c7 45 e8 0f 00 00 00    movl   $0xf,-0x18(%rbp)
 91a:   c7 45 ec 0a 00 00 00    movl   $0xa,-0x14(%rbp)
 921:   48 8d 35 8d 01 00 00    lea    0x18d(%rip),%rsi        # ab5 <_ZStL19piecewise_construct+0x1>
 928:   48 8d 3d f1 06 20 00    lea    0x2006f1(%rip),%rdi        # 201020 <_ZSt4cout@@GLIBCXX_3.4>
 92f:   e8 7c fe ff ff          callq  7b0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@p
lt>
 934:   48 89 c2                mov    %rax,%rdx
 937:   8b 45 e8                mov    -0x18(%rbp),%eax
 93a:   89 c6                   mov    %eax,%esi
 93c:   48 89 d7                mov    %rdx,%rdi
 93f:   e8 9c fe ff ff          callq  7e0 <_ZNSolsEi@plt>
 944:   48 8d 35 6f 01 00 00    lea    0x16f(%rip),%rsi        # aba <_ZStL19piecewise_construct+0x6>
 94b:   48 89 c7                mov    %rax,%rdi
 94e:   e8 5d fe ff ff          callq  7b0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@p
lt>
 953:   48 89 c2                mov    %rax,%rdx
 956:   8b 45 ec                mov    -0x14(%rbp),%eax
 959:   89 c6                   mov    %eax,%esi
 95b:   48 89 d7                mov    %rdx,%rdi
 95e:   e8 7d fe ff ff          callq  7e0 <_ZNSolsEi@plt>
 963:   48 89 c2                mov    %rax,%rdx
 966:   48 8b 05 63 06 20 00    mov    0x200663(%rip),%rax        # 200fd0 <_ZSt4endlIcSt11char_trait
sIcEERSt13basic_ostreamIT_T0_ES6_@GLIBCXX_3.4>
 96d:   48 89 c6                mov    %rax,%rsi
 970:   48 89 d7                mov    %rdx,%rdi
 973:   e8 48 fe ff ff          callq  7c0 <_ZNSolsEPFRSoS_E@plt>
 978:   8b 45 e8                mov    -0x18(%rbp),%eax
 97b:   8b 55 ec                mov    -0x14(%rbp),%edx
 97e:   89 d3                   mov    %edx,%ebx
 980:   01 d8                   add    %ebx,%eax
 982:   89 45 e8                mov    %eax,-0x18(%rbp)
 985:   48 8d 35 35 01 00 00    lea    0x135(%rip),%rsi        # ac1 <_ZStL19piecewise_construct+0xd>
 98c:   48 8d 3d 8d 06 20 00    lea    0x20068d(%rip),%rdi        # 201020 <_ZSt4cout@@GLIBCXX_3.4>
 993:   e8 18 fe ff ff          callq  7b0 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@p
lt>
 998:   48 89 c2                mov    %rax,%rdx
 99b:   8b 45 e8                mov    -0x18(%rbp),%eax
 99e:   89 c6                   mov    %eax,%esi
 9a0:   48 89 d7                mov    %rdx,%rdi
 9a3:   e8 38 fe ff ff          callq  7e0 <_ZNSolsEi@plt>
 9a8:   48 89 c2                mov    %rax,%rdx
 9ab:   48 8b 05 1e 06 20 00    mov    0x20061e(%rip),%rax        # 200fd0 <_ZSt4endlIcSt11char_trait
sIcEERSt13basic_ostreamIT_T0_ES6_@GLIBCXX_3.4>
 9b2:   48 89 c6                mov    %rax,%rsi
 9b5:   48 89 d7                mov    %rdx,%rdi
 9b8:   e8 03 fe ff ff          callq  7c0 <_ZNSolsEPFRSoS_E@plt>
 9bd:   b8 00 00 00 00          mov    $0x0,%eax
 9c2:   48 83 c4 18             add    $0x18,%rsp
 9c6:   5b                      pop    %rbx
```

EPIC
Institute of Technology
Powered by epam

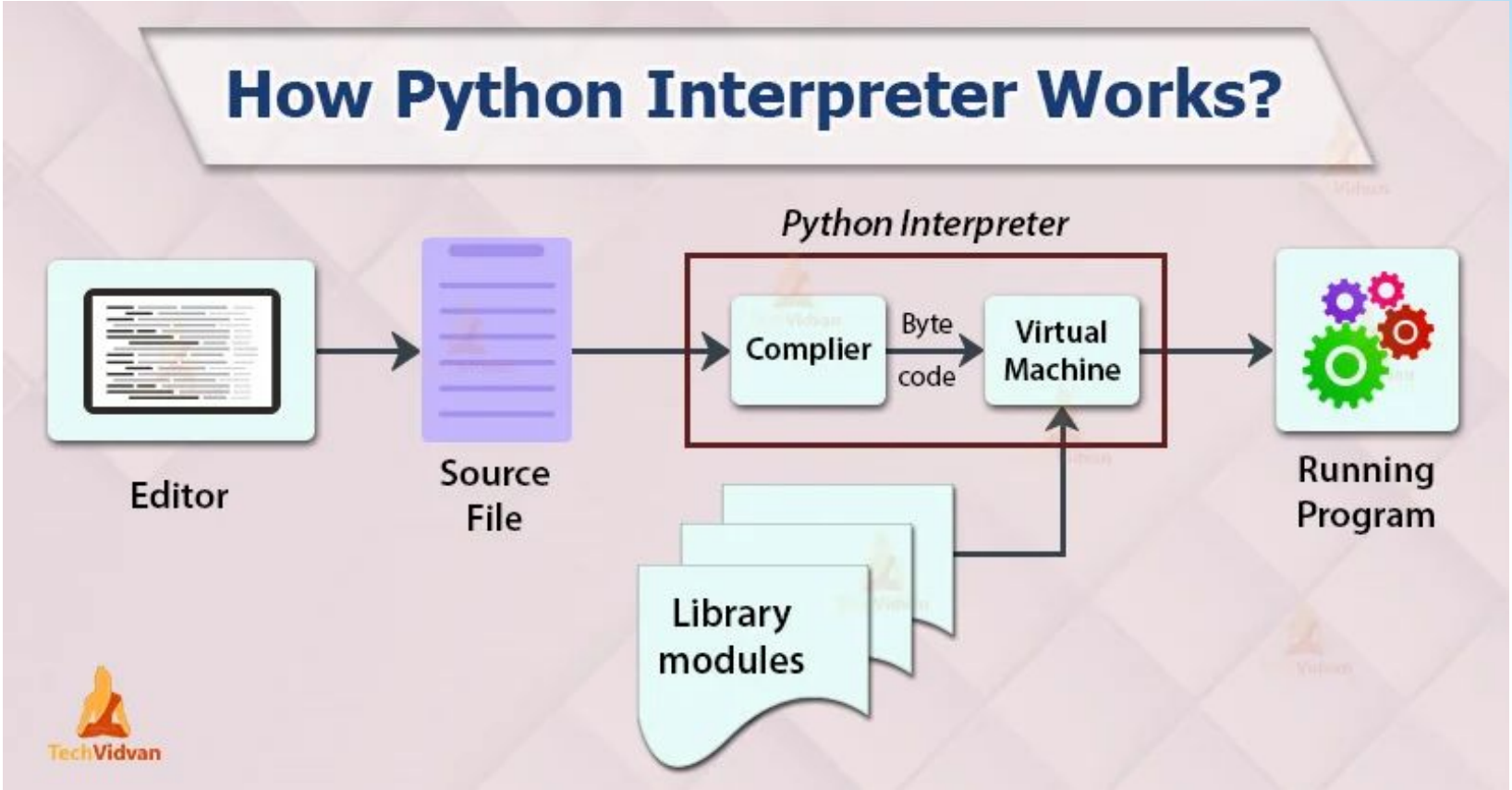
Inline assembler

```
1  use std::arch::asm;
2
3  fn main() {
4      let x: u64 = 15;
5      let y: u64 = 10;
6      let mut sum: u64;
7      // compare order with last slide
8      // another asm
9      unsafe {
10         asm!(
11             "add {y}, {x}",      // y = y + x
12             "mov {sum}, {y}",    // sum = y
13             x = in(reg) x,       // readonly x to asm
14             y = in(reg) y,       // readonly y to asm
15             sum = out(reg) sum, //sum - output only
16         );
17     }
18     println!("{}", sum);
19 }
```

```
Compiling playground v0.0.1 (/playground)
 Finished dev [unoptimized + debuginfo] target(s) in 0.59s
  Running `target/debug/playground`
```

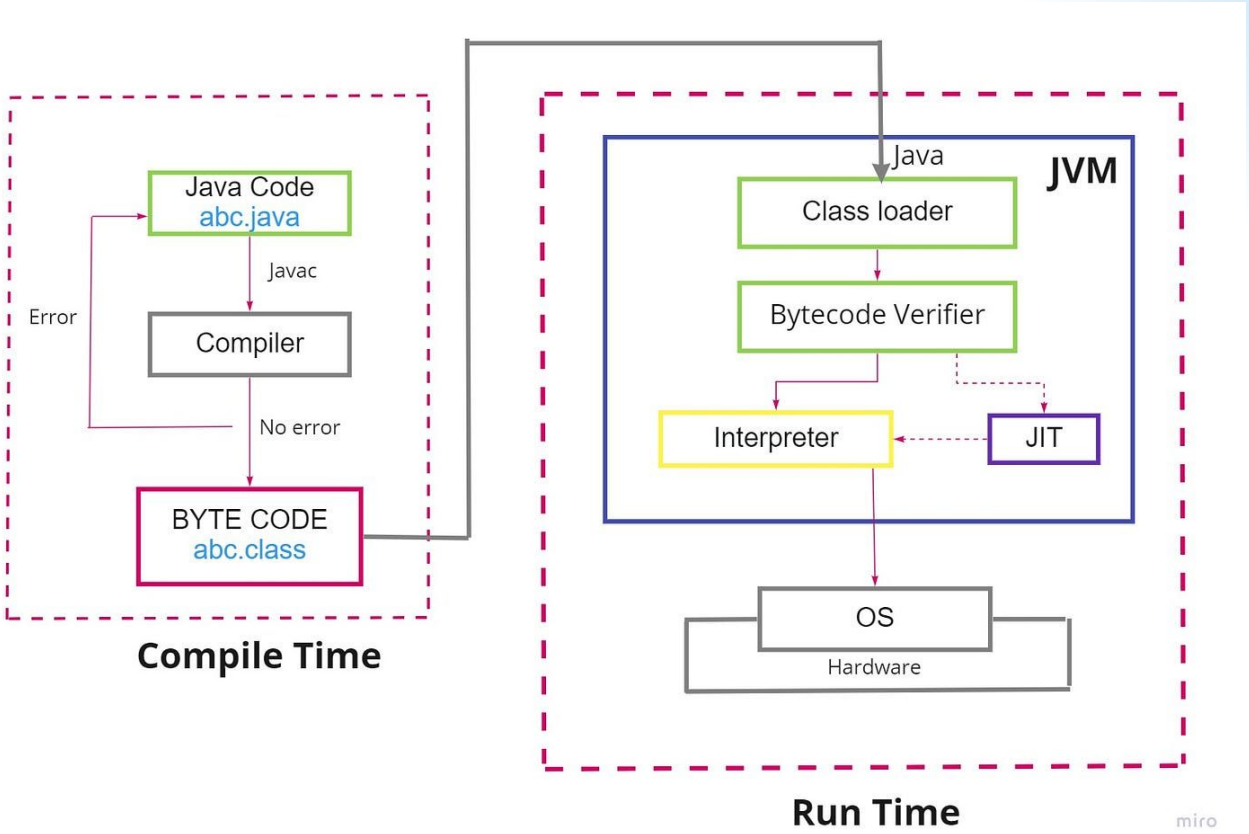
EPIC
Institute of Technology
Powered by epam

Interpreter



EPIC
Institute of Technology
Powered by epam

JIT



EPIC
Institute of Technology
Powered by epam

That's All Folks!