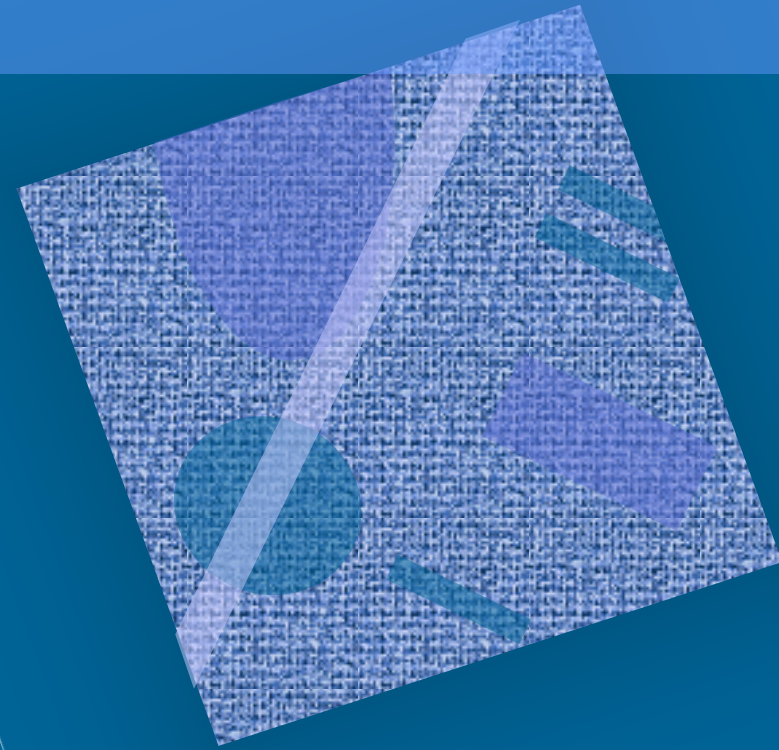


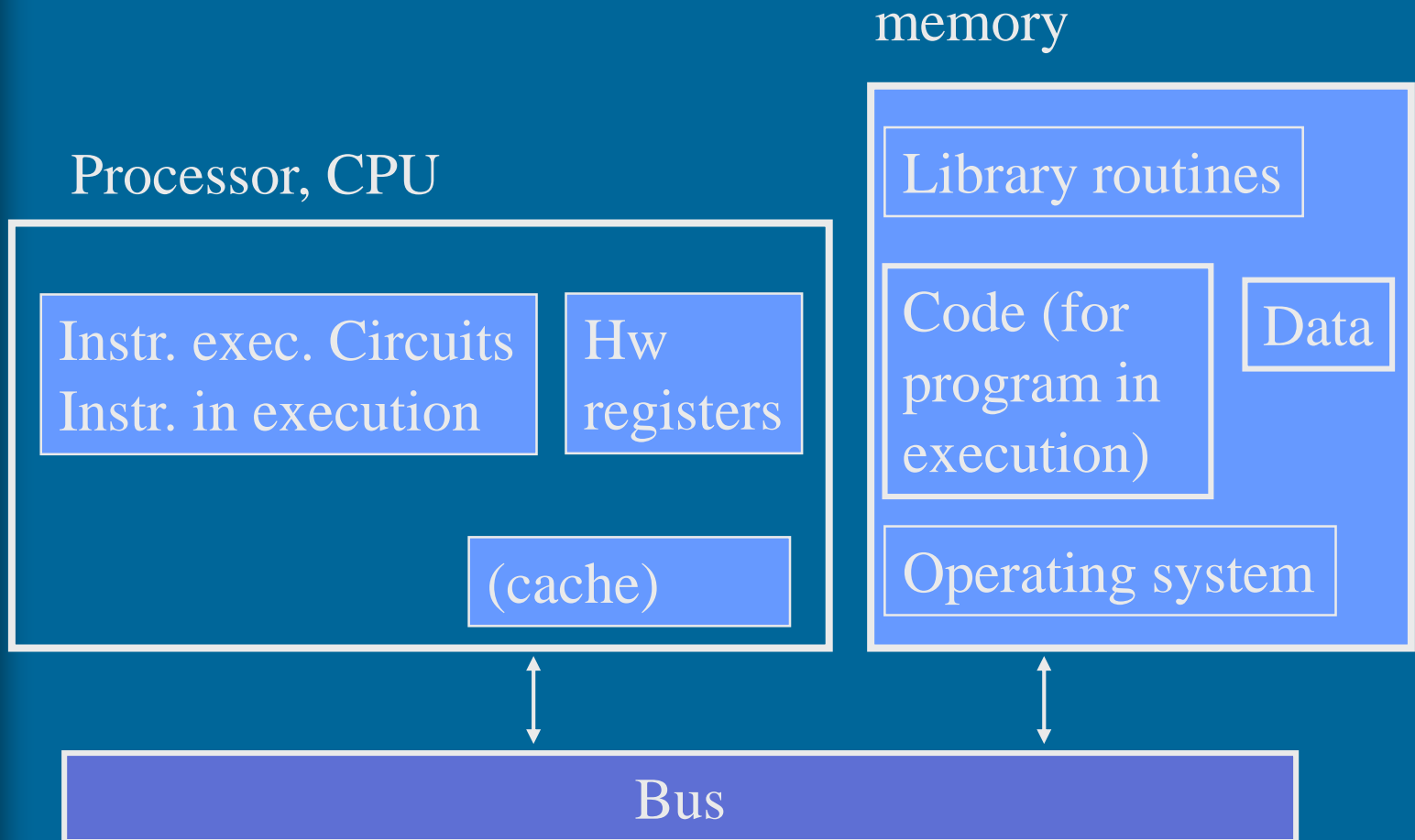
# Lecture 5

# Processor and Bus



Fetch-Execute Cycle  
Processor States  
Exceptions and Interrupts  
Program Placement in Memory

# Execution Time Contents of Processor and Memory



# Processor Operation

PC=0 init value

start

IR = Instruction Register

PC = Program Counter

käskeyjen nouto- ja suoritusssykli

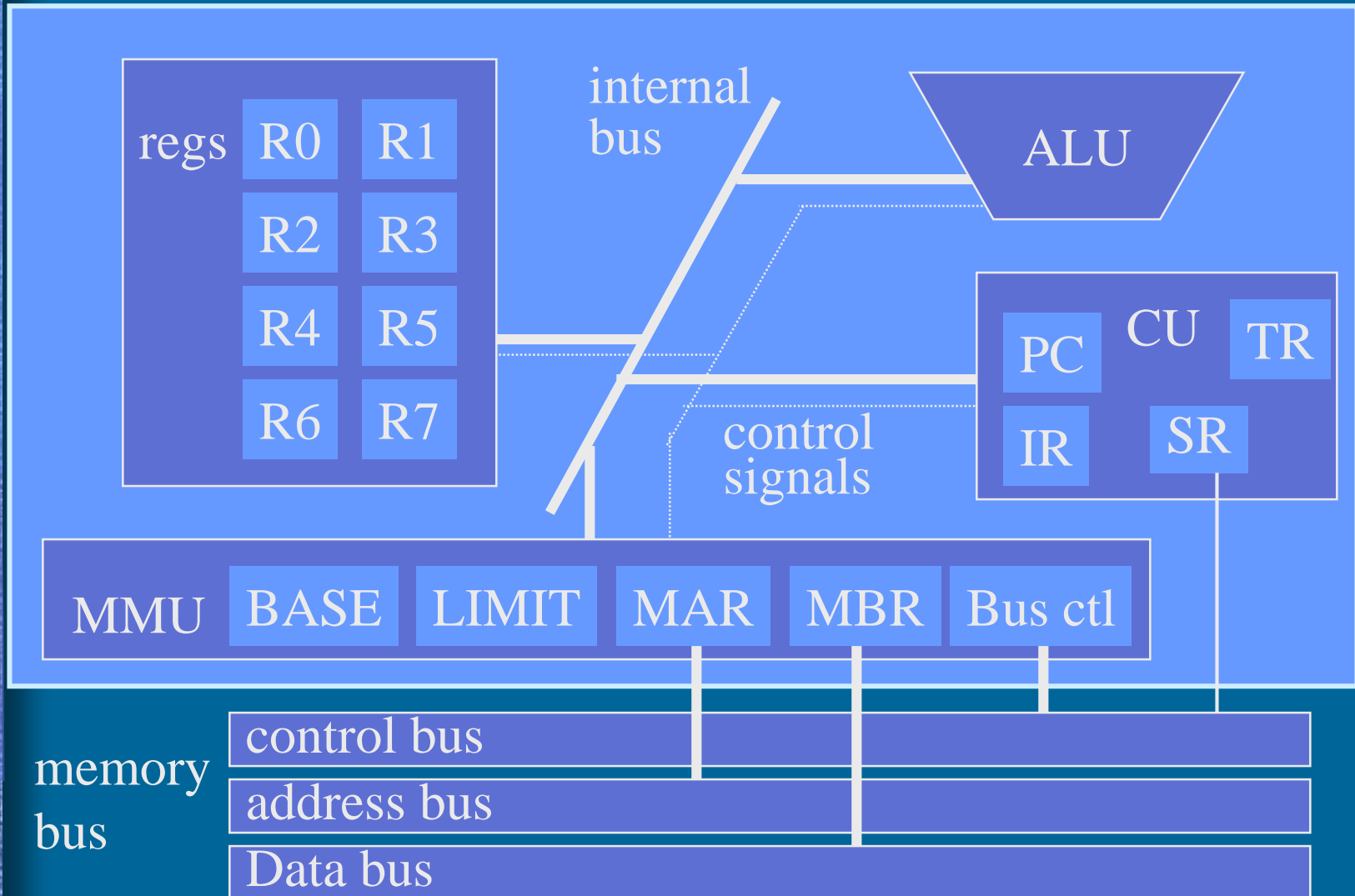
Instr fetch  
 $IR \leftarrow \text{mem}(PC)$

Increment Program Counter  
 $PC \leftarrow PC+1$

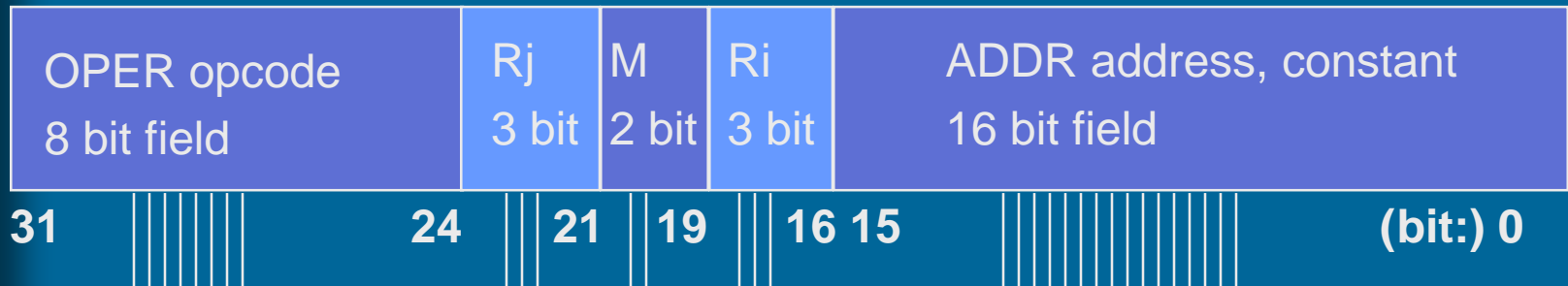
Execute instruction in IR  
(may modify PC)  
(may cause mem references)

instruction fetch-execute cycle

# TTK-91 Processor Structure



# Break Instruction to Fields, and Compute Effective Address (EA)



- Fields can be read with special wires from instr. reg (IR)
- Compute EA, result to TR
  - Rj: 1st operand and result reg
  - if Ri=0, then  $TR \leftarrow ADDR$
  - else  $TR \leftarrow (Ri)+ADDR$ 
    - Do addition in ALU (or some special circuit)
    - If  $ADDR = 0$ , then  $TR \leftarrow (Ri)$
  - Now, Effective Address (EA) is in TR
- You can use contents of TR as is, as address of data in memory, or as indirect address of data in memory
  - 2nd operand definition, use M to decide on how TR data is used

# Instruction Execution

- Titokone visualization shows phases in simulation
- Store R4, @10(R1) ; R1 =20, R4=15
  - Instruction fetch and incrementing PC
    - PC → MAR, “bus read”, wait for “mem → MBR”
    - MBR → IR
    - PC+1 → PC
      - E.g., PC → ALU1, 1 → ALU2, +, wait, ALU → PC
      - E.g., PC → INCin, wait, INCout → PC (own circuit)
  - Instruction execution phase
    - R1 → ALU1, 10 → ALU2, +, (odota), ALU → TR
    - TMP → MAR, ”bus read”, wait for “mem→MBR”  
MBR → MAR, R4 → MBR, ”bus write”, wait for “MBR→mem”
  - Total 3 memory references (this instruction)

# TTK-91 Machine Code

Opcode	Rj	M	Ri	Attribute (constant, addr)
8 b	3 b	2 b	3 b	16 b

- Each instruction is 32 bits
- Each instruction has opcode
- How to interpret registers and attribute, depends on opcode and mode (M)
- Data types:
  - 32-bit integer, or raw 32-bit values
  - No floating points, characters, booleans, etc

# TTK-91 registers

- 8 general registers (for any type of use)
  - Only these registers can be directly referenced (with read or write ops) in machine instructions
  - All calculations (all work) happen with these registers
    - only 8 "memory slots" for actual work
  - R0 work register
    - If index register  $R_i=0$ , it denotes value 0, and not contents of index register  $R_i$ , i.e., "no indexing"
  - R1-R5 work or index registers
    - Type depends on where in instr register is used in
    - work reg 1st operand, index reg in 2nd operand
  - Stack Pointer SP (i.e., R6)
  - Frame Pointer FP (i.e., R7)

Stack Pointer

Frame Pointer

To implement subroutines (not for ordinary computational work)



# TTK-91 Control Unit (CU)

- PC - Program Counter (Instruction Pointer, IP)
  - Address of next (not current) instruction to execute
- IR - Instruction Register
  - Current instruction in execution
- TR - Temporary Register
  - Extra storage for data needed for instruction execution
- SR - State Register
  - Current processor state and limitations

# TTK-91 State Register SR

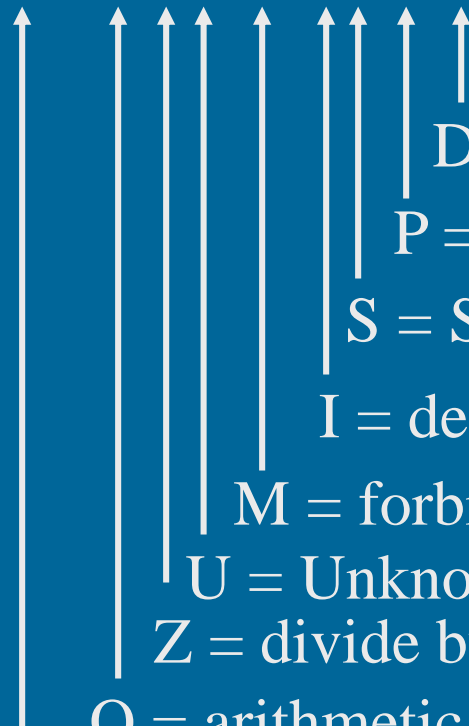
- State info on what happened at the processor when this or previous instruction(s) was executed
  - Errors, exceptions, interrupts
  - Machine instruction was supervisor call (SVC)
  - Results of comparisons
- State info on what has happened in the system recently
  - Device signals (device interrupts) not yet processed
- State info on what the processor is allowed to do
  - Privileged or normal execution state?
    - All memory and all instructions allowed, or not?
  - Interrupt processing allowed or not?

# Ttk-91 State Register SR <sup>(9)</sup>

32 bits (each has value 0 or 1)

SR:

GEL OZUM IS P D ????????



D = Interrupts Disabled (*kesk. estetty*)

P = Privileged mode (*etuoik. tila*)

S = SVC (supervisor call) *palvelupyöntö*

I = device Interrupt (*laitekeskeyty*s)

M = forbidden Memory address

U = Unknown instruction

Z = divide by Zero

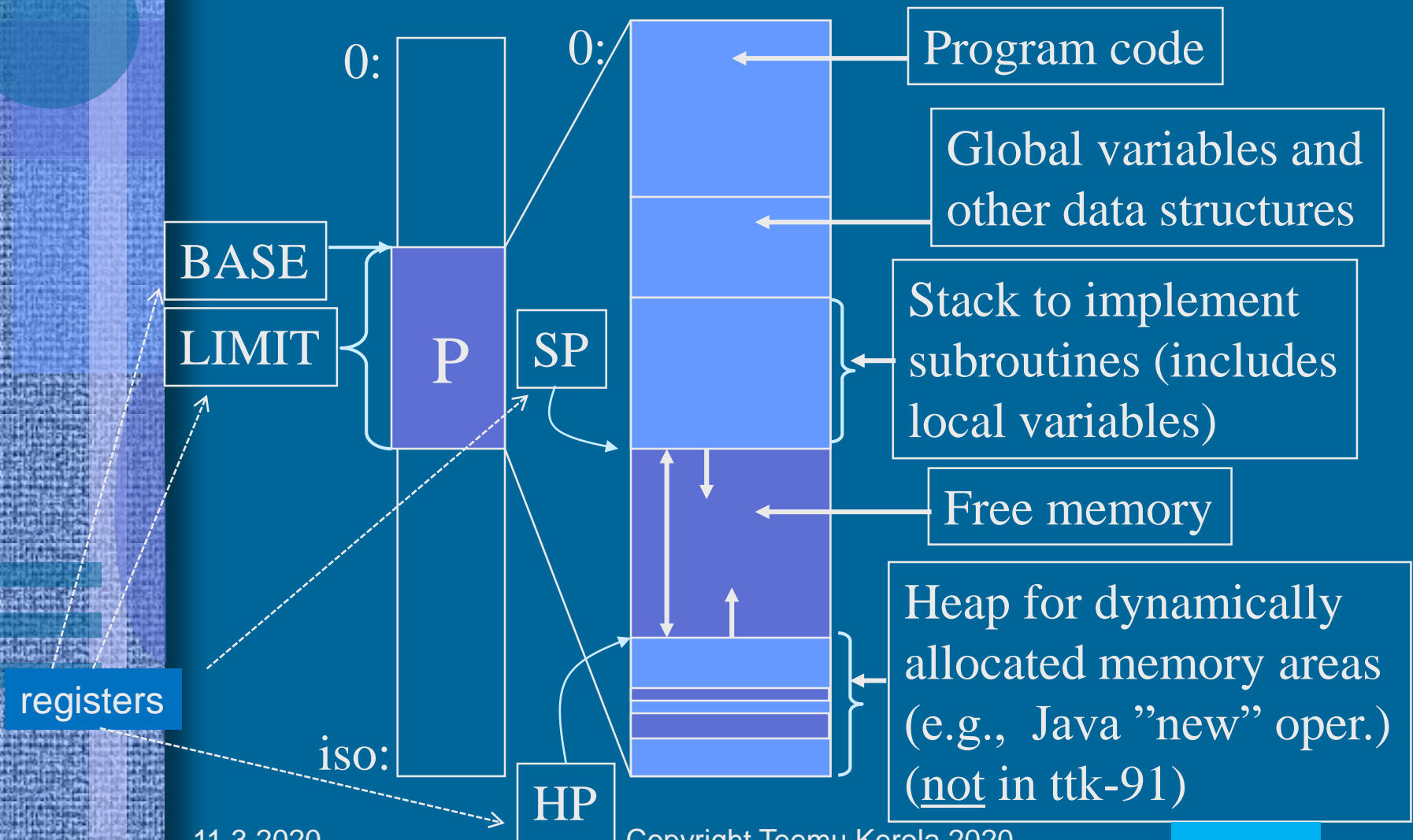
O = arithmetic Overflow

GEL = comparison indicators: Greater, Equal, Less

# Memory Use for Program P

Physical memory

Virtual mem for P



# Address Translation Mechanisms for Virtual Memory

- Based on base and limit register pairs
  - ttk-91, 8086, ...
- Paging
  - Page tables
  - Address space divided into same size "pages"
- Segmenting
  - Address space divided into (large?) variable size "segments"
    - Code segment, data segment, literal area, heap, stack, ...

More  
info?



OS  
course

More  
info?



OS  
course

# Interrupt Processing

- Interrupts fetch-execute cycle, "surprise subroutine call", moves control to operating system
- Every possible interrupt type is previously known, i.e., nothing really surprising does not happen!
- For each interrupt type there is specific interrupt handler (subroutine) in the operating system
- At the end of every fetch-execute cycle the HW checks for any interrupt existence from SR, and branches to its interrupt handler when needed.
  - Old PC and SR saved, new PC and SR set
  - E.g., interrupt type 3:  $PC \leftarrow 3$  tai  $PC \leftarrow \text{mem}(3)$ 
    - physical memory address 3?
  - Interrupt processing is sometimes disabled (SR bit D in ttk-91)
  - Return from interrupt handler with some special instruction (e.g., IRET or "return-from-interrupt-handler")
    - Recover old PC and old SR

# Processor Operation

PC=0 init value

start

IR = Instruction Register

PC = Program Counter

Instr fetch

$IR \leftarrow \text{mem}(PC)$

Check for interrupts  
(may modify PC)

instruction  
fetch-execute cycle

Increment  
Program Counter  
 $PC \leftarrow PC+1$

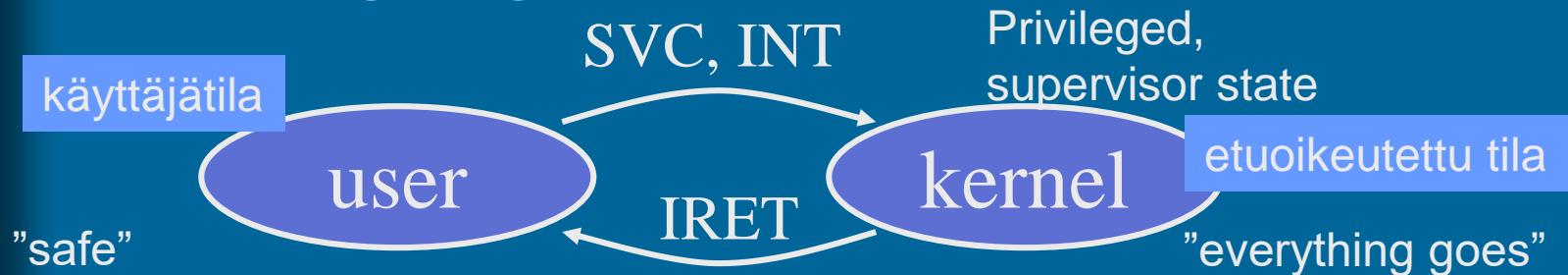
Execute instruction in IR  
(may modify PC)  
(may cause mem references)

# Interrupt Handlers

- Important part of OS (Operating System)
- Before control moves to interrupt handler, processor state is set to privileged state (supervisor state, kernel mode)
  - SR bit P is on (1) → processor is in privileged state
  - In kernel mode the OS can reference all of memory (MMU: BASE=0, LIMIT="very big")
  - In kernel mode the OS can use all instructions
    - E.g., IRET, ClearCache, ReadBASE, SetBASE, ReadLIMIT, SetLIMIT, SetD, ResetD, ReadSR, ...
- When returning from interrupt handler processor state is returned back to the one it was originally
  - including registers BASE and LIMIT, bit P in SR, etc



# Changing CPU Execution State



- User state → Privileged state
  - Interrupt or direct supervisor call (SVC instruction)
  - Interrupt handler checks whether state change ok
- Privileged state → User state
  - Privileged machine instruction “return from interrupt handler”, e.g., IRET (Pentium II)
  - Returns control and processor state to those before control was transferred to interrupt handler

keskeytykäsittelijä

# Titokone - TTK-91 simulator

- Ordinary program written in Java
- TTK-91 processor/system components as data structures
  - registers, MMU, CU, memory
- Simulate fetch-execute cycle, one instr. at a time
- Includes also parts of operating system
  - assembler, loader, debugger, interrupt handlers
- Graphical user interface (UI)

See Processor.java in Titokone code:

titokone.jar\fi\hu\cs\titokone\Processor.java

(<http://www.cs.helsinki.fi/group/nodes/kurssit/tito/Processor.java.txt> )

# TTK-91 Fetch-execute Cycle

Fetch instruction from simulated memory

$$\text{IR} = \text{mem}[\text{PC}]$$

Break instr. into fields (OPER, Rj, M, Ri, ADDR) and compute initial address to TR ( $\text{ADDR} + \text{reg}[\text{Ri}] + \text{ADDR}$ )

$$\text{ADDR} = \text{IR} \bmod 32768 \quad \text{TR} = \text{reg}[\text{Ri}] + \text{ADDR}$$

Do enough memory references (M) to get 2nd operand to register TR

$$\text{TR} = \text{mem}[\text{TR}]$$

Select proper simulation code based on opcode (OPER)

if (opcodeOK[OPER] = FALSE) then SR.U = 1;

Simulate instruction execution effects into registers (R0...R7, SR, PC, MAR, MBR)

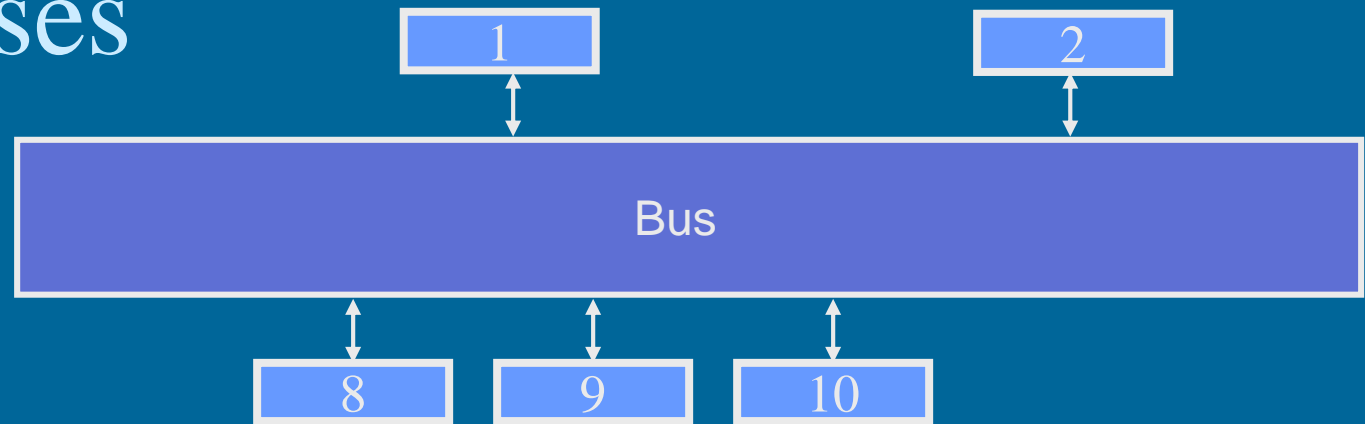
$$\text{ADD Rj, M ADDR(Ri)} \Rightarrow \text{reg}[\text{Rj}] += \text{TR};$$

Stop cycle if SVC or interrupt

$$\text{SR.O} = \dots$$

Simulator in C: <http://www.cs.helsinki.fi/group/nodes/kurssit/tito/simu/simu.c>

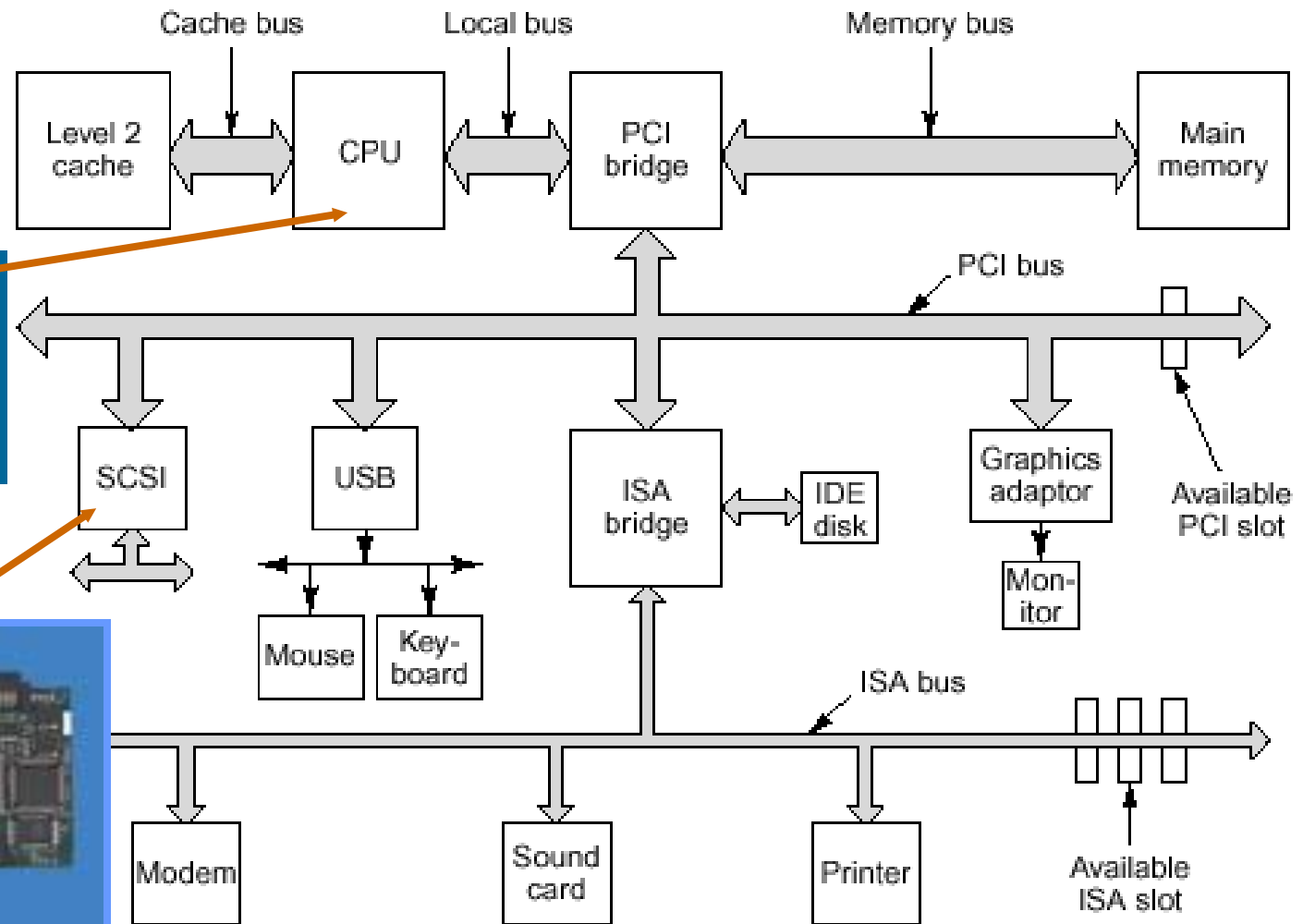
# Buses



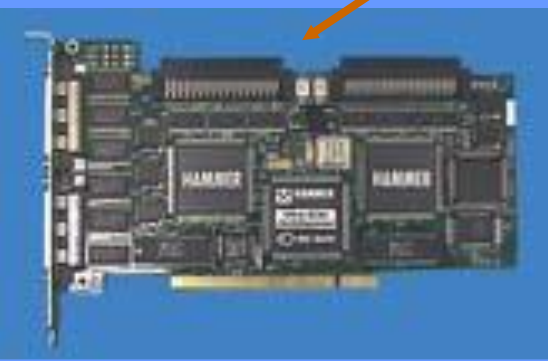
- Each device on the bus has simple address
- One transmits, all listen, only the "correct" device will receive and react to it
- Many different buses, hierarchy of buses
- Those close to CPU are faster

# Bus hierarchy

Typical mother board for Pentium II system



In its own chip, with level 1 cache



PCI to SCSI bridge

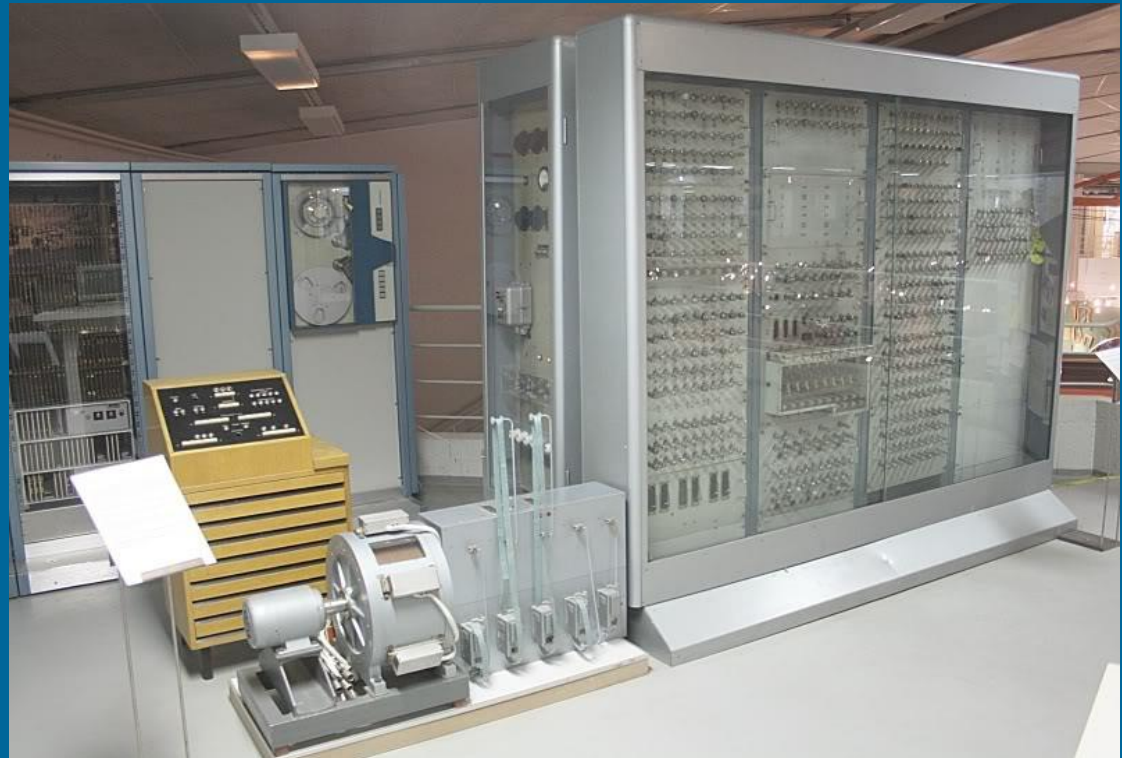
Fig. 3-50 [Tane99]

-- End --

(Tekniikan museo)

(The museum of technology)

ESKO, 1960.  
First "computer"  
built in Finland,  
out of date from  
the start.  
20 additions per  
second.  
Good learning  
experience.



- Program code read from paper tape (10 readers)
  - Code and data were not in the same memory!
- Subroutine call implemented with control transfer to another tape reader (with subroutine code in looped tape)