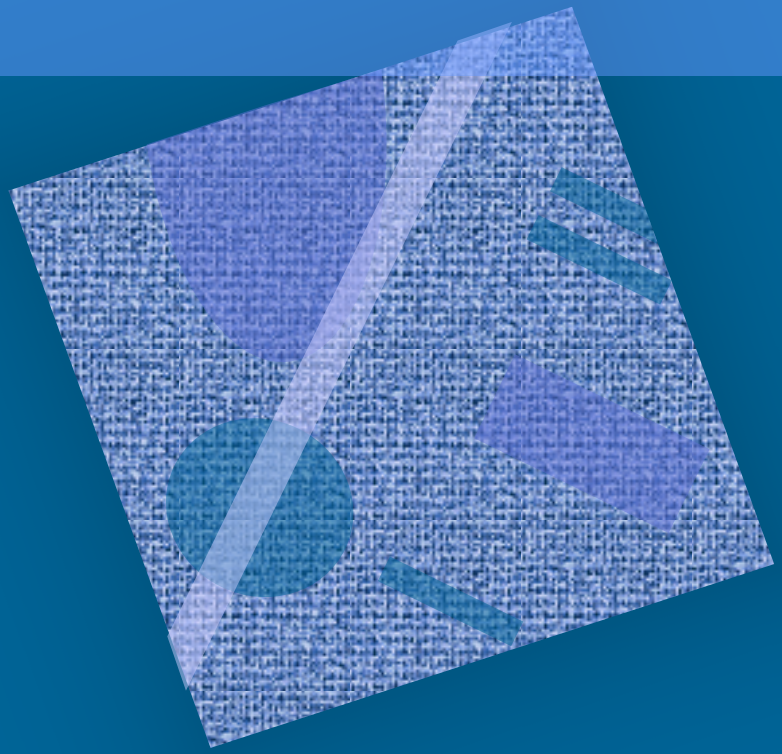
Lecture 2

Ttk-91 System



Ttk-91 hardware

Location of data

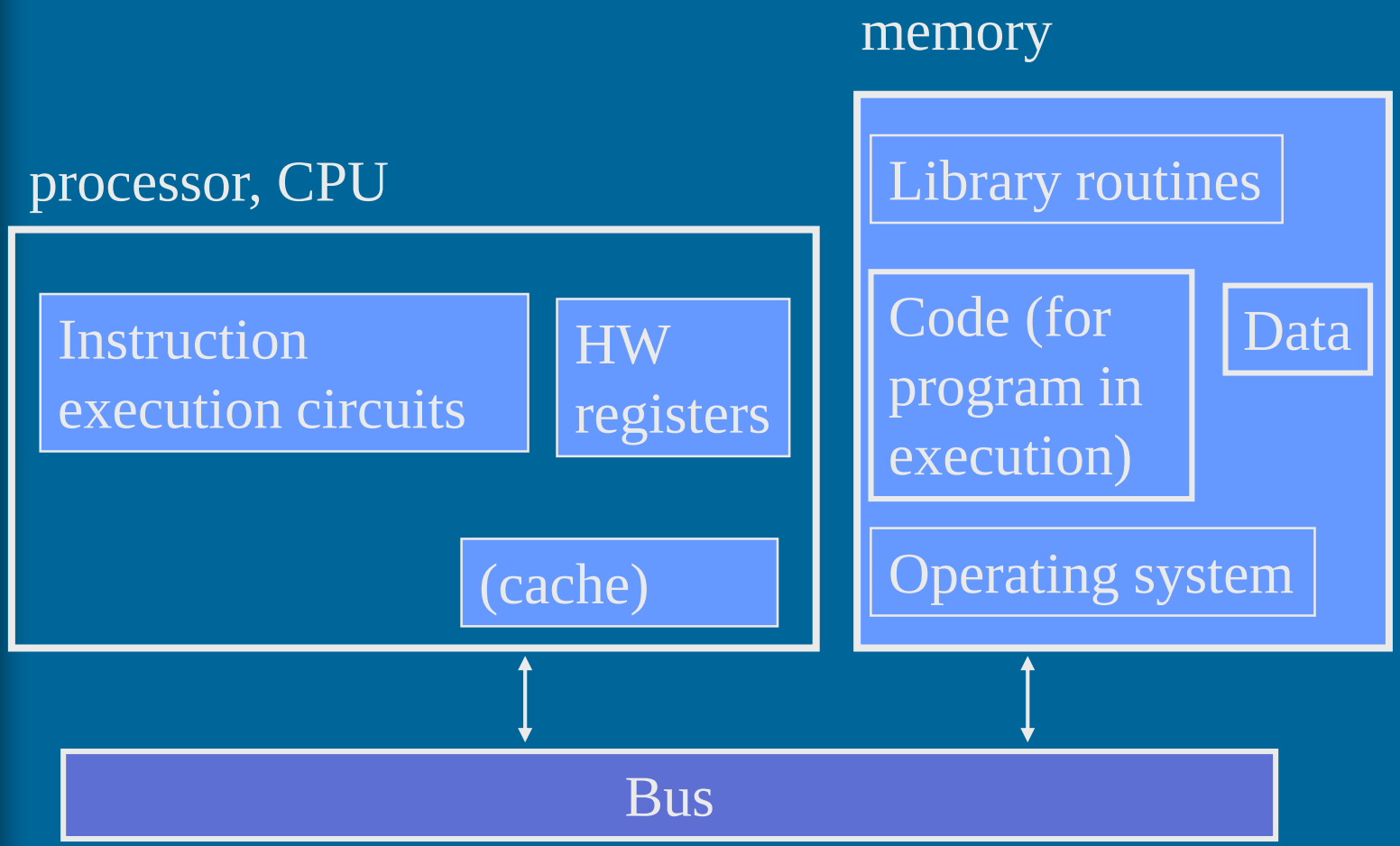
Memory use

Ttk-91 machine language

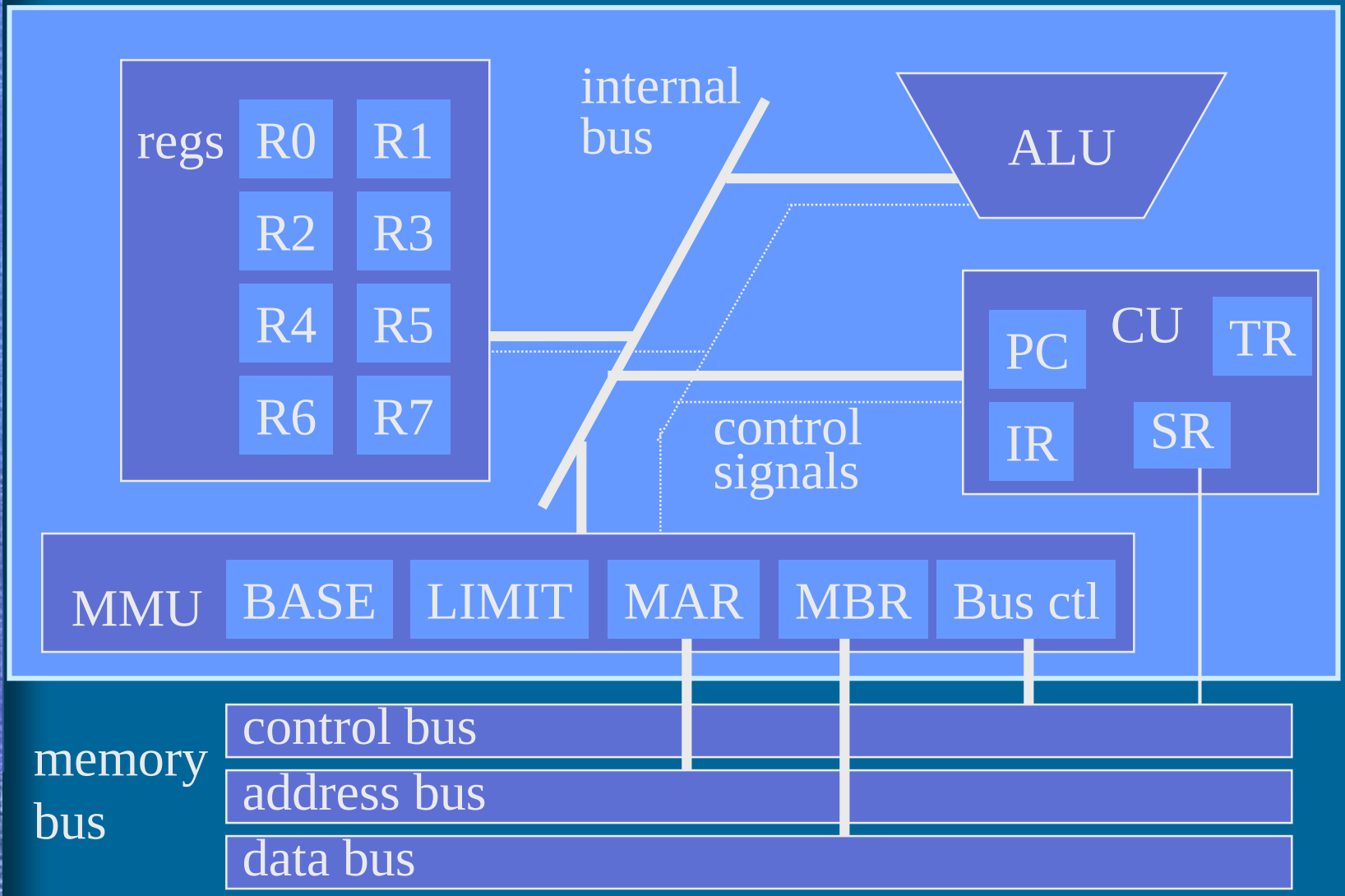
Addressing data

Indexing, arrays, records

Execution Time Contents of Processor and Memory



TTK-91 Processor Structure



TTK-91 Machine Instruction

- Ttk-91 instruction in bit-level is always:

(small) memory address or constant

(addressing mode)

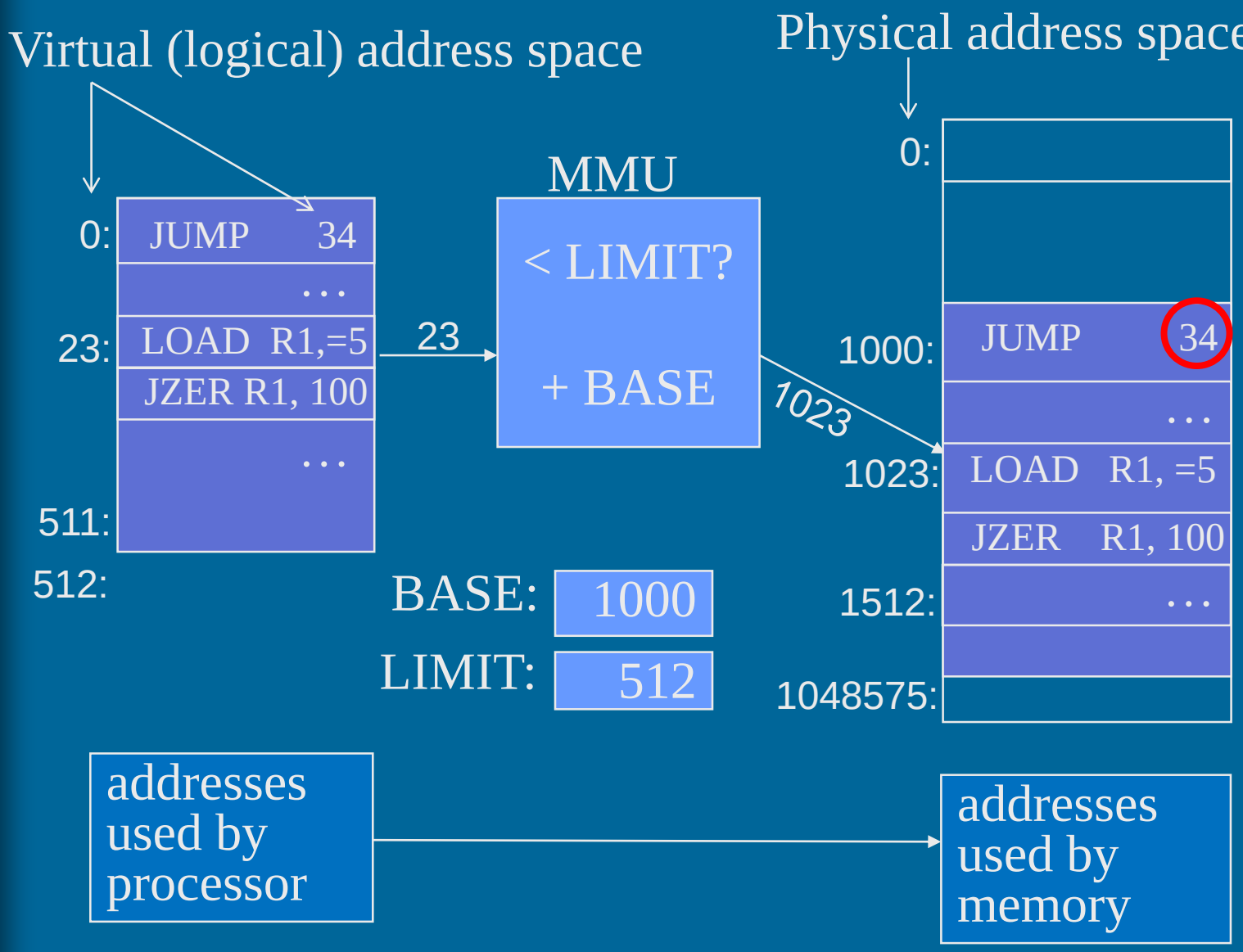
Rj = 1st operand, result

Ri = index register (0 ≡ not used now)

M = nr of memory loads for 2nd operand
(before operation execution)

- 00, immediate operand (STORE: direct addressing)
- 01, direct addressing (STORE: indirect addressing)
- 10, indirect addressing (STORE: invalid value)
- (11, invalid value → error, bad mode exception)

TTK-91 Base and Limit Registers



Execution Time Data Location

- In memory (in main memory)
  - large — E.g., 256 MB, or 64 M 32 bit words
  - slow — Esim. 50-150 ns
  - in data area, or in constant field in instruction?
- In register set
  - small — E.g., 256 B, or 64 words of 32 bits
  - fast — E.g., 1 ns

TTK-91: R0-R7 + PC + ...

When in value of variable X in memory?
When is it in register?
Where in memory? How is it referenced?

Discuss

TTK-91 Operations

- Addressing memory
  - Ordinary: load & store alone, or with arithmetics
  - Stack operations (to implement subroutines)
- I/O instructions for predefined I/O devices
- Integer arithmetic operations (integer data)
- Logical ops for bit values (raw bit data)
- Shift ops for bit values (raw bit data)
- Control transfer ops
  - Where is the next instruction to execute?
    (unless it is in the default position, next mem location)
- Other instructions (e.g., NOP)

TTK-91 Memory Addressing Ops

- LOAD
  - Same instruction is also used to copy register values (”move” operation)
- STORE
  - Always stores in memory
- PUSH, POP, PUSHR, POPR
  - To implement subroutines
  - More details given later on

```
LOAD R1, X
LOAD R5, @ptrX
LOAD R0, R5
```

```
STORE  R2, X
STORE  R3, Tbl(R4)
```

```
PUSH  SP, R1   ; store to stack
POP     SP, R1   ; take from stack
```

TTK-91 I/O Operations

- IN

```
IN      R3, =KBD
```

  - Read (integer) value to register from given device (only device KBD defined)
- OUT

```
OUT     R2, =CRT
```

  - Output (integr) value in register to given device (only device CRT defined)
- Devices?
  - KBD - KeyBoarD, stdin
  - CRT – Cathode Ray Tube, display, stdout
  - Nothing else! (no disks, no networks, …)

TTK-91
Integer Ops

- LOAD (”move”)
- ADD, SUB
- MUL
- DIV, MOD

```
LOAD  R3, R1  ; R3 ← R1
```

```
ADD  R3,  X   ; R3 ← R3+ Mem(X)
```

```
SUB  R3, =1   ; R3 ← R3-1
```

```
MUL  R3, Tbl(R1)   ; R3 ← R3 * Mem(Tbl+R1)
```

```
LOAD  R1,=14
DIV     R1,=3          ; R1 ← 4
LOAD  R1,=14
MOD    R1,=3           ; R1 ← 2
```


Discuss

TTK-91
Logical (Bit) Ops (4)

- NOT, AND, OR, XOR
  - For all 32 bits in word
  - Operations are done one bit at a time (bitwise)

```
LOAD  R1, =12      ; R1 = 000...000 1100
LOAD  R2, =5       ; R2 = 000...000 0101
```

```
AND   R1, R2       ; R1 = 000...000 0100
OR    R1, R2       ; R1 = 000...000 1101
XOR   R1, R2       ; R1 = 000...000 1001
NOT   R1           ; R1 = 111...111 0011
```

TTK-91 Bit Shifts

- SHL, SHR, SHRA
  - Move bits left or right
  - Fill with zeroes (or with sign bit (leftmost bit), SHRA)

```
LOAD R1,=5    ; R1 = 000…000 00101 = 5
SHL  R1,=1    ; R1 = 000…000 01010 = 10
```

  - Shifting one bit left is the same as multiplying by 2. (if leftmost sign bit remains the same all the time)
  - With positive numbers, shifting right is usually the same as divide by 2. Much faster op than divide!

```
LOAD R1,=5    ; R1 = 000…000 00101 = 5
SHR  R1,=1    ; R1 = 000…000 00010 = 2
```

```
LOAD R1,=-5   ; R1 = 111…111 11011 = -5
SHRA R1,=1    ; R1 = 111…111 11101 = -3
```

TTK-91 Control Transfer Ops

kontrollinsiirto

- JUMP
  ```
  JUMP   Loop
  ```
- COMP
  ```
  COMP   R3, =27
  ```
  ```
  COMP   R2, X
  ```
  - Set comparison result to status register SR: L, E or G
- JLES, JEQU, JGRE, JNLE, JNEQU, JNGRE
  - Based on comparison result stored in state register SR (e.g., preceding COMP instruction)
  ```
  JGRE   Loop
  ```
- JNEG, JZER, JPOS, JNNEG, JNZER, JNPOS
  - Based on value of 1st register Rj
  ```
  JPOS   R1, Loop
  ```
- CALL, EXIT (subroutines, discussed later)
- SVC
  ```
  SVC   SP, =HALT   ; program completes exec.
  ```

TTK-91 Other Instructions

- NOP

```
NOP
```

  - No OPeration, empty instruction, do nothing
  - Reserves 1 word of memory (in code)
  - Executes the same way as any other instruction
    - Uses time
    - May have address to jump to

```
        jzer  r4, pass
        ....
pass    nop
        ....
```

Data Addressing in TTK-91

- Only for the 2nd operand
  - 1st operand is always a register

välitön operandi

- Immediate operand (no memory access)
  - OPER Rj, =ADDR(Ri) M = 0 = 0b00
  - 2nd operand: ADDR+Ri
  - Either component can be missing (ADDR=0, Ri=R0)

(bit presentation)

- Direct addressing (indexed addressing)
  - OPER Rj, ADDR (Ri) M = 1 = 0b01
  - 2nd operand : Mem(ADDR+Ri)

suora muisti-osoitus

- Indirect addressing (indirect indexed addressing)
  - OPER Rj, @ADDR(Ri) M = 2 = 0b10
  - 2nd operand : Mem(Mem(ADDR+Ri))

epäsuora muisti-osoitus

indeksoitu muisti-osoitus

Get 2nd Operand with Indexing

```
LOAD R4,=Tbl(R3)
LOAD R4,Tbl(R3)
LOAD R4,@Tbl(R3)
```

- Compute 1st so called effective address, EA

EA= Tbl + (R3) = 201

- Do as many memory accesses (or none) as specified by mode
  - ”=”: M=0 — R4 ← 201 (immediate)
  - nothing: M=1 — R4 ← Mem[201] (= 11) (indexed direct)
  - ”@”: M=2 — R4 ← Mem[Mem[201]] = Mem[11] = 300 (indexed indirect)

Just register nr after ’@’ ⇒ M=1, just 1 mem access (feature, not a bug)

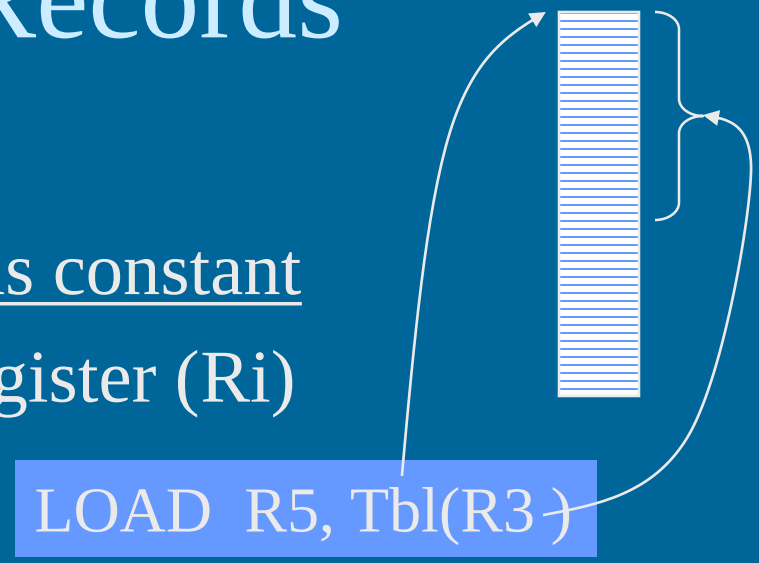
```
LOAD R4,@R4
```

STORE instr ⇒ 1 less mem reads and one mem write

discuss

Indexed Addressing Mode with Arrays and Records

- Arrays (1D)
  - Array starting address as constant
  - Array index in index register (Ri)



- Records (objects)
  - Record starting address in register
  - Record field's relative address (in record) as instruction constant

tietueet

TTK-91 Assembler Pseudo-Instructions (Pragmas)

pseudokäskyt, kääntäjän ohjauskäskyt

- Do not generate executable instructions
  - Directives to compiler, or loader

```
Hundr    EQU  100
```

- EQU - Equal
  - Give certain value to symbol, in symbol table

```
LOAD R1, =Hundr
```

- DC - data constant
  - Reserve one worrd from memory, initialize it to given value, set it's address as value for given symbol
  - Allocate space for variable or constant

```
Fifty    DC   50
```

```
LOAD R1, Fifty
```

- DS - data segment
  - Reserve many words from memory, initial values undefined, set it's address as value for given symbol
  - Allocate space for array or record

```
Tbl      DS   200
```

```
LOAD R3,Tbl(R1)
```


Discuss

TTK-91 Symbolic Assembly Language Program

symbo-linen konekieli

hello.k91

```
X       DC      13
Y       DC      15

MAIN    LOAD    R1, X
        ADD     R1, Y
        OUT     R1, =CRT
        SVC     SP, =HALT
```

What are the values for these symbols?

X? MAIN? CRT? ADD?

https://www.cs.helsinki.fi/group/titokone/v1.100/kayttoohje/manual.html

-- End --

Some typical 80x86 instructions and their function

Instruction	Function
JE name	If equal (CC) EIP = name}; EIP − 128 ≤ name < EIP + 128
JMP name	{EIP = NAME};
CALL name	SP = SP − 4; M[SP] = EIP + 5; EIP = name;
MOVW EBX,[EDI + 45]	EBX = M [EDI + 45]
PUSH ESI	SP = SP − 4; M[SP] = ESI
POP EDI	EDI = M[SP]; SP = SP + 4
ADD EAX,#6765	EAX = EAX + 6765
TEST EDX,#42	Set condition codea (flags) with EDX & 42
MOVSL	M[EDI] = M[ESI]; EDI = EDI + 4; ESI = ESI + 4

Fig. 3.32 [Patterson-Hennessy 1998]