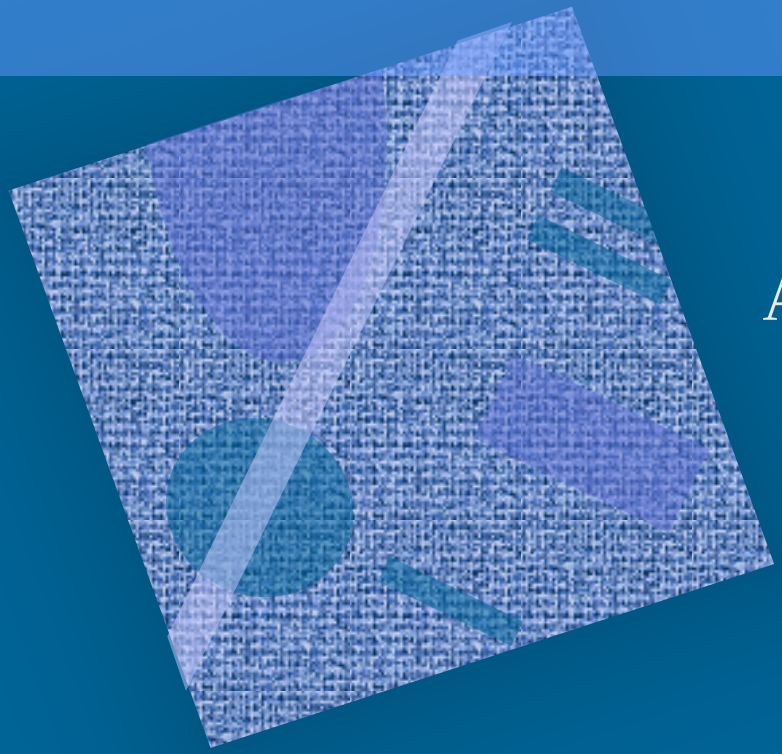
Lecture 3

Ttk-91 Assembly Language Programming

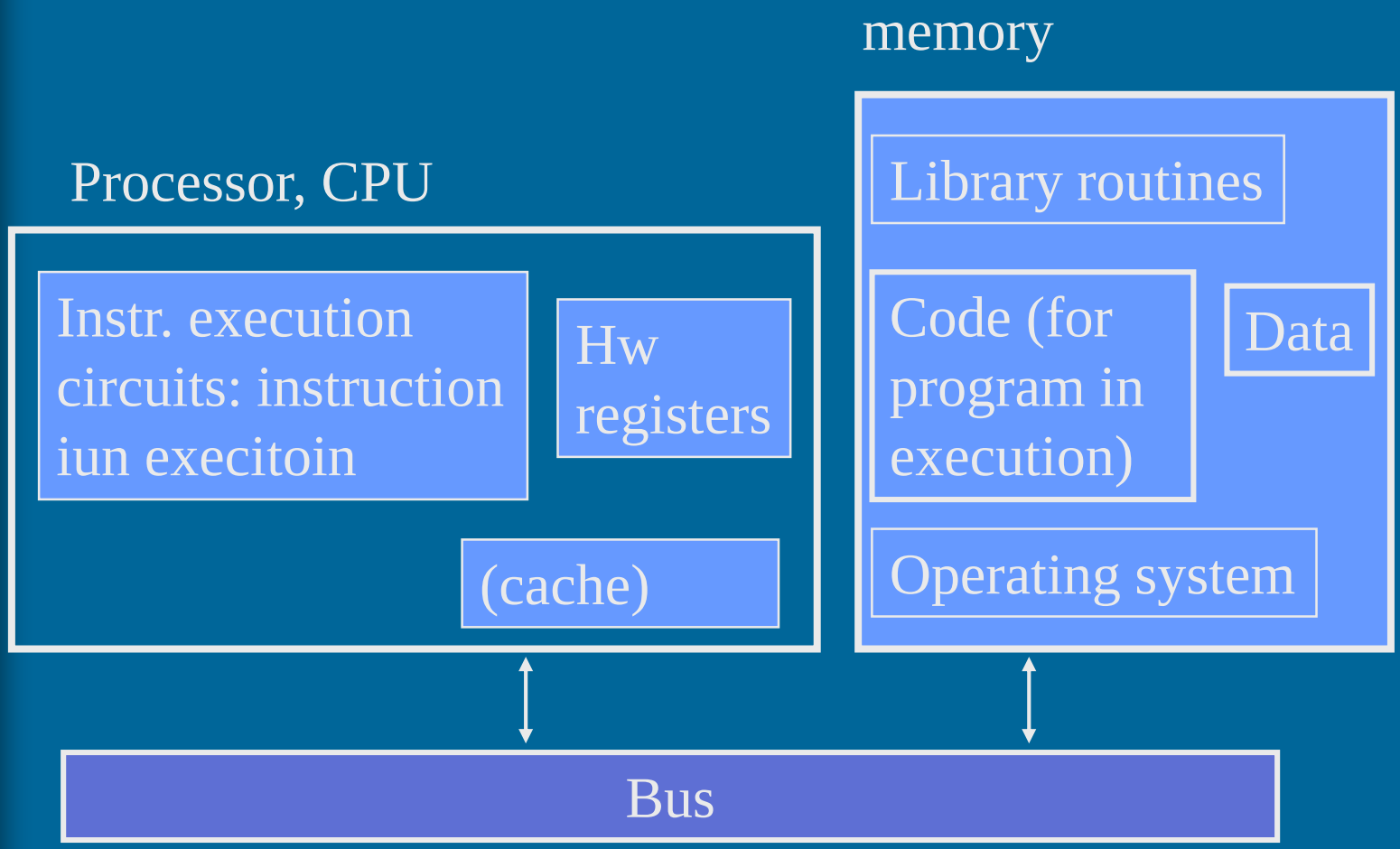


Program representation

Assembly Language with ttk-91

(Titokone, TitoTrainer)

Execution Time Contents of Processor and Memory



Program Representation in Machine Code

- Machine language instruction set defines instruction set architecture (ISA) for system
- Program in machine language representation in memory (TTK-91)

	Machine language instruction in memory	As number
0:	0000 0010 000 00 000 0000 0000 0110 0100	DEC: 33554532
1:	0000 0010 001 01 000 0000 0000 0110 0100	DEC: 36175972
2:	0001 0100 001 00 000 0000 0000 0000 0000	DEC: 337641472
3:	0010 0010 000 00 000 0000 0000 0000 0110	HEX: 22000006
4:	0000 0001 001 00 000 0000 0000 1110 0100	HEX: 012000E4
5:	0000 0000 000 00 000 0000 0000 0000 0000	HEX: 00000000

- opcode - Rj M Ri - ADDR: constant -

Program Representation in Symbolic Assembly (Machine) Language

- Symbolic assembly language instruction
  - Instruction given in fields (parts)
  - Some field values are given as symbols
  - Easier for humans to read and write

Symb asmbly lang	
LOAD R2, =100	
LOAD R1, 100	
DIV R1, R2	
JZER 6	JZER Loop
STORE R1, X	X≡228
NOP	

Assembly lang, machine lang				
0000 0010	010	00	000	0000 0000 0110 0100
0000 0010	001	01	000	0000 0000 0110 0100
0001 0100	001	00	010	0000 0000 0000 0000
0010 0010	000	00	000	0000 0000 0000 0110
0000 0001	001	00	000	0000 0000 1110 0100
0000 0000	000	00	000	0000 0000 0000 0000

Discuss

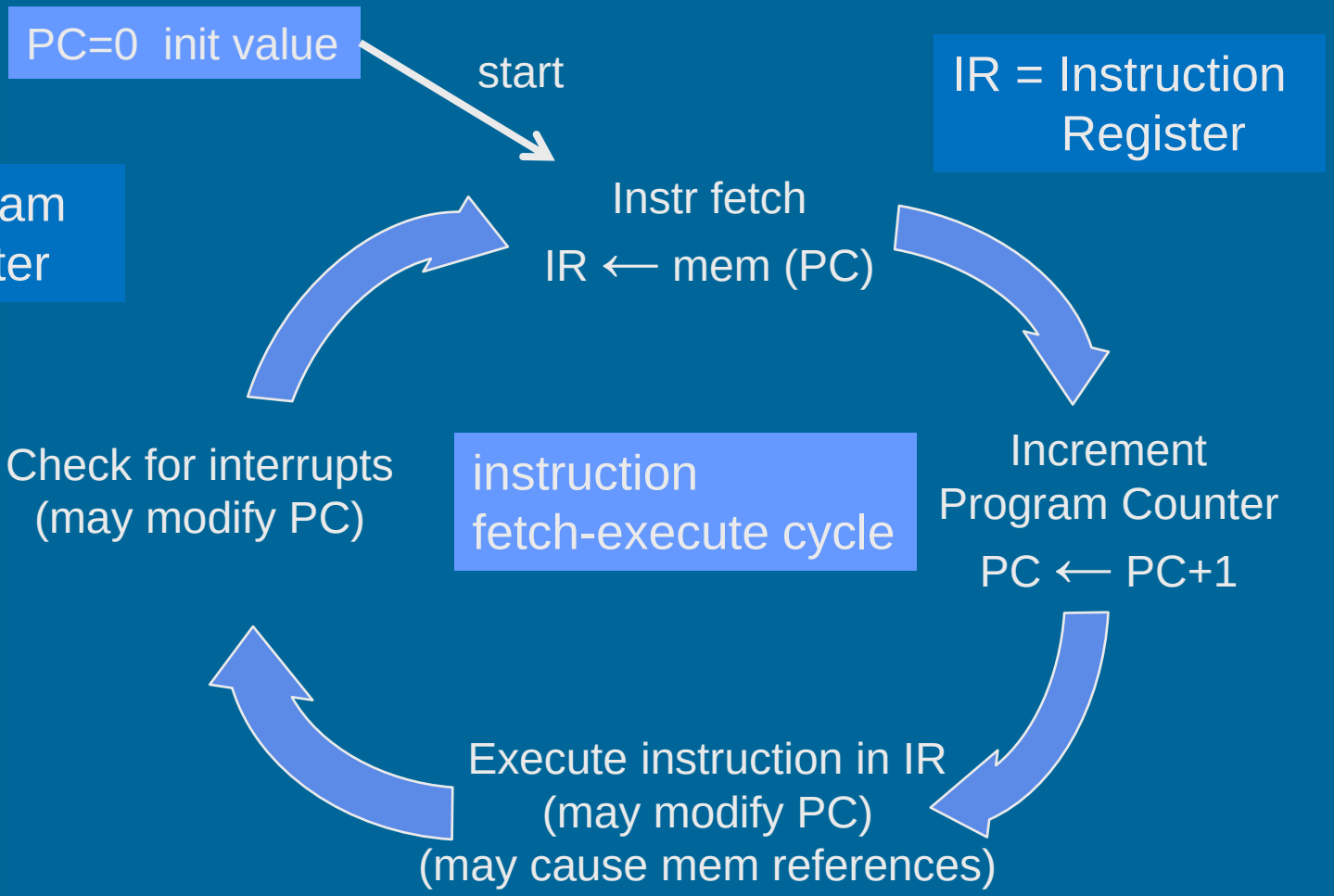
TTK-91 Machine Code

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

Discuss

Processor Operation



Location of Data at Execution Time

- Register (fastest)
  - Compiler usually decides, when data is kept in register
- Cache (fast)
  - HW decides automatically for data referenced recently
- Memory (slow)
  - Compiler/loader decides location in memory
    - Global data areas reserved at program load (in data area)
    - Some constants are in machine instructions (in code area)
  - Program reserves space from stack during execution
    - Subroutine local data structures and parameters in stack
  - OS (or run time library) reserves space at execution time
    - Dynamic data in heap (e.g., Java *new* operation)
- Disk, file server (too slow, not possible)
  - Data must be copied to memory before referencing

How to Reference Data?

- Data in memory
  - Address with (logical) memory address (e.g., 0x6F123456 or 3459321)
  - Address with symbol (e.g., PersId or X) when symbolic asmbly lang used
    - Symbol value is memory address or (relative) field location
    - PersId ≡ 0x6F123456, X ≡ 3459321
      (hexadecimal) (decimal)
- Data in cache
  - Address the same way as if the data would be in memory
  - At reference time (during instruction execution) you do not know whether data is in cache or in memory, or how long it takes to reference the data!
  - Speed somewhere between register and memory reference speeds
  - There can be many caches: L1 (small, fastest), L2 (bigger), L3 (large)
- Data in register
  - Address with reg. nr (e.g., nr 6 or 18) (in ttk-91: 0-7)
  - Use reg names: R3, FP, F5, PS, SR, I2, etc. (ttk-91: R0-R7, SP, FP)
- Data as constant in instruction (in IR at exec time)
  - No addressing needed, by default there is only one such location

Data and its address

```
X  DC     12
   ....
   LOAD  R1, =X
   LOAD  R2, X
```

int x =12;

Address of variable X is value of symbol X

Value for symbol X (address of variable X)

Value of variable X

memory

230
12345
12556
128765
12222
12
12998

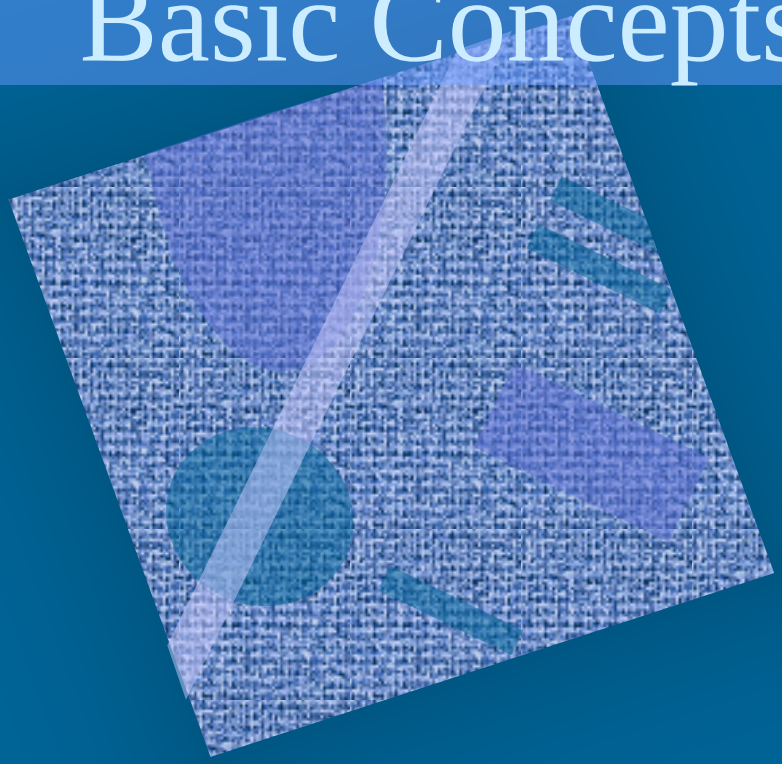
X ≡ 230:

- Address of variable X is 230
- Value of variable X is 12
- Value of symbol X is 230
  - Symbols exist usually only at compilation time
  - For error messages, symbol table is sometimes kept also at run time

symb.table

```
…
X   230
…
```

Ttk-91 and Basic Concepts in Programming



Arithmetics

Data structures

Control, branching, loops

Complex data structures

Basic Concepts in Programming

- Arithmetic statement
  - How to perform arithmetic operatoipns?
- Simple data structures
  - Memory data reference mode supports directly
  - 1-dimensional arrays, records, objects
- Control – where is next instruction to execute?
  - Selection: if-then-else, case
  - Repeat: for loops, while loops
  - Subroutines (lecture 4), errors and other interrupts
- Complex data structures
  - Lists, multi-dimensional arrays, etc
  - First compute address of data, then do the reference

Arithmetic statement (3)

Reserve space for (global) variables

```
A       DC      0
B       DC      0
C       DC      0
```

```
int  a, b, c;
…
b = 34;



a = b + 5 * c;
```

code

```
        LOAD    R1, =34
        STORE   R1, B
        ....
        LOAD    R1, B
        LOAD    R2, C
        MUL     R2, =5
        ADD     R1, R2
        STORE   R1, A
```

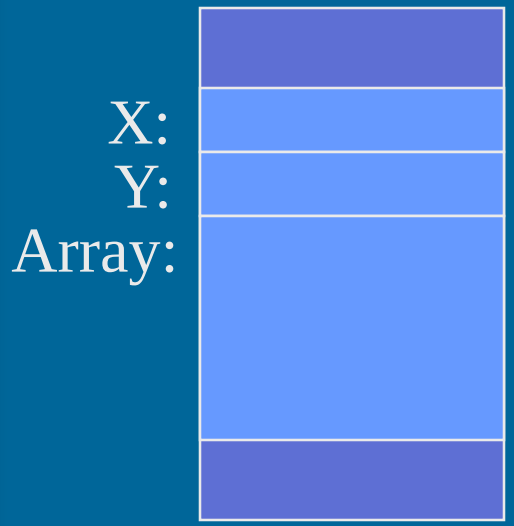
or:

```
LOAD R1,=5
MUL  R1,C
ADD  R1, B
STORE R1,A
```

Allocating Space and Use of (Everywhere Visible) Global 1-dimensional Array

```
int X, Y;
int Array[30];
…
X = 5;
Y = Array[X];
```

```
X       DC      0
Y       DC      0
Array   DS      30
        …
        LOAD    R1, =5
        STORE   R1, X
        LOAD    R1, X
        LOAD    R2, Array(R1)
        STORE   R2, Y
```



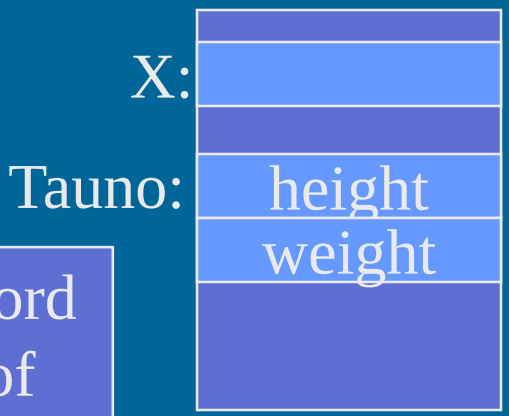
Optimizing compiler could leave out this ”LOAD R1, X” instr.

Allocating Space and Use of Global record (3)

```
int X;
struct Tauno {
        int Height;
        int Weight;
        }
…
X = Tauno.Weight
```

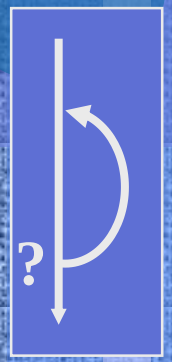
Relative address of field ”Weight” inside record Tauno

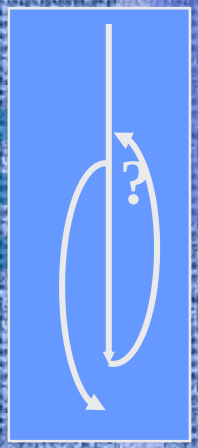
```
X       DC      0
Tauno   DS      2
Height  EQU     0
Weight  EQU     1
…
LOAD R1,=Tauno
LOAD R2, Weight(R1)
STORE R2, X
```



Address of record
Is the address of
Its 1st word (byte)

For statement






```
for (int i=20; i < 50; ++i)
    T[i] = 0;
```

```
T       DS      50
I       DC      0
        …
        LOAD  R1, =20        init
        STORE R1, I

Loop    LOAD  R2, =0         body
        LOAD  R1, I
        STORE R2,
T(R1)

        LOAD  R1, I          incre-
        ADD   R1, =1         ment
        STORE R1, I

        LOAD  R3, I          test
        COMP  R3, =50
        JLES  Loop
```

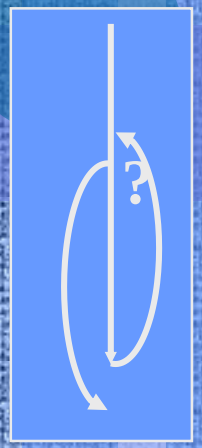
Would it be better
to keep value of i
in register? Why? When?

What is the value of i
At the end? Does it exist?

What if different
loop semantics?

Discuss

While-do statement



```
X = 14325;
Xlog = 1;
Y = 10;
while (Y < X) {
        Xlog++;
        Y = 10*Y
        }
```

```
        LOAD   R1, =14325
        STORE  R1, X
        LOAD   R1, =1   ; R1=Xlog
        LOAD   R2, =10  ; R2=Y
While   COMP   R2, X
        JNLES  Done
        ADD    R1, =1
        MUL    R2, =10
        JUMP   While
Done    STORE  R1, Xlog ; save result
        STORE  R2, Y
```

What is useful to keep in memory?

What is useful to kepp in which register and when?

X in R3?

Discuss

Array Index Checking

```
for (int i=20; i < 50; ++i)
        T[i] = 0;
```

```
I       DC    0

T       DS    50  ; data
Tsize   DC    50  ; size
        …
```

Can you combine loop control and index checking? Optimizing compiler can!

```
        LOAD  R1, =20
        STORE R1, I
Loop    LOAD  R2, =0
        LOAD  R1, I
        JNNEG R1, ok1
        SVC   SP,=BadIndex
ok1     COMP  R1, Tsize
        JLES  ok2
        SVC   SP, =BadIndex
ok2     STORE R2, T(R1)
        LOAD  R1, I
        ADD   R1, =1
        STORE R1, I ; 50 OK!
        LOAD  R3, I
        COMP  R3, =50
        JLES  Loop
```

Multidimensional Arrays

T:

34	**57**	**76**
21	76	23
24	56	876
54	75	777

T:

T[0][0]=34
T[0][1]=57
T[0][2]=76
T[1][0]=21
T[1][1]
T[1][2]
T[2][0]
T[...][...]

- Store row-wise
  - C, Pascal, Java?
- Store column-wise
  - Fortran

T:

T[0][0]=34
T[1][0]=21
T[2][0]=24
T[3][0]=54
T[0][1]=57
T[1][1]
T[2][1]
T[...][...]

- Other ways exist, e.g.,
  - Each row allocated separately
  - T[i] is address of row i
- 3 or more dimensions
  - similarly!

Referencing Complex Data Structures

```
int T[10, 20];
```

```
T     ds  200
Trows equ 10
Tcols  equ 20
```

T[i,j] = 34; ???

- First compute the relative address of referenced data within the data structure
- Do the reference with simple indexing
  - Same way as for 1-dimensional array

Row-wise

```
Load r1, i
Mul   r1,  =Tcols
Add   r1, j

Load  r2,=34
Store r2, T(r1)
```

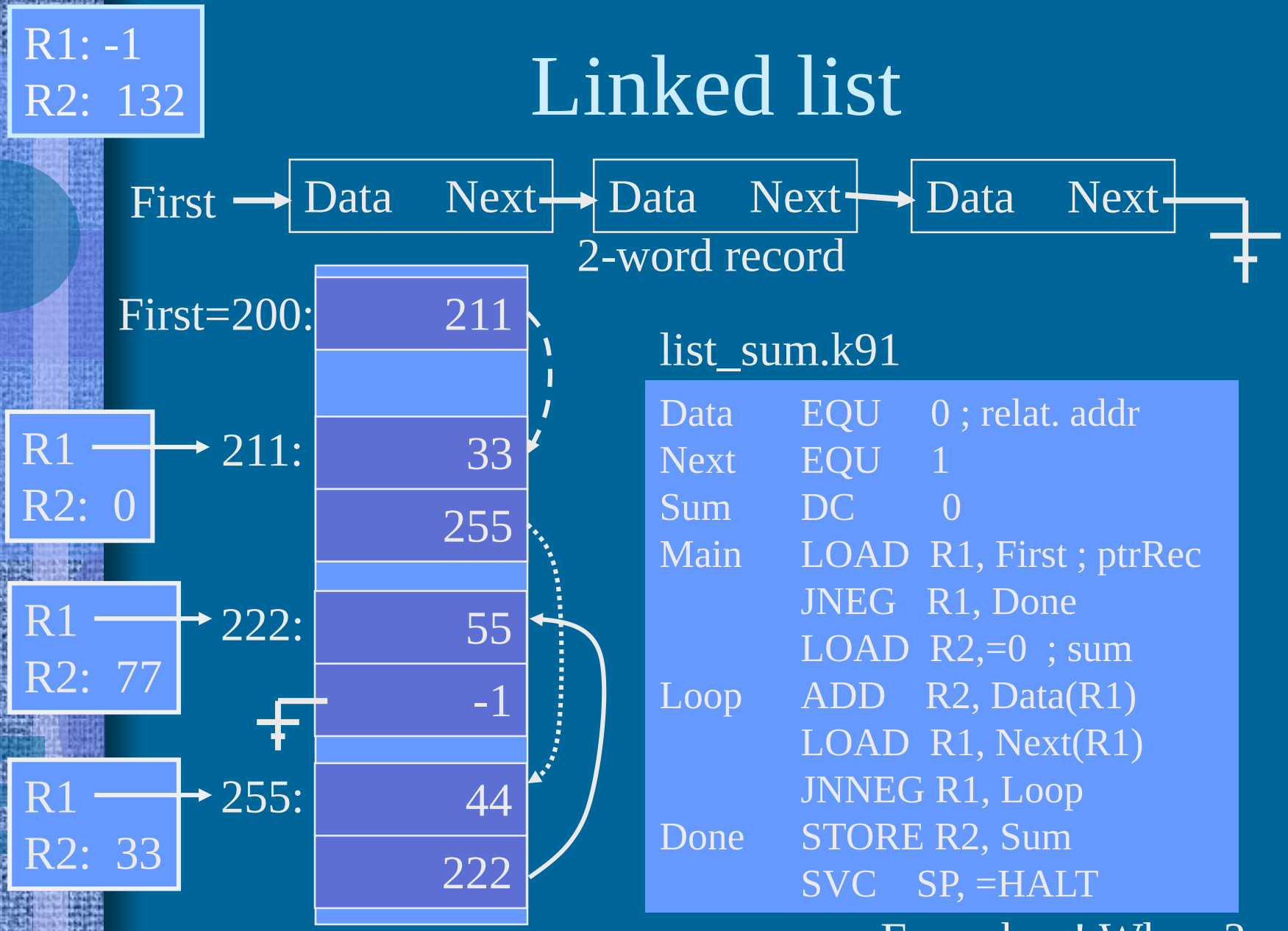
Column-wise

```
Load r1, j
Mul   r1,  =Trows
Add  r1, i

Load  r2,=34
Store r2, T(r1)
```


Discuss

Linked list



list_sum.k91

```
Data    EQU   0 ; relat. addr
Next    EQU   1
Sum     DC    0
Main    LOAD  R1, First ; ptrRec
        JNEG  R1, Done
        LOAD  R2,=0  ; sum
Loop    ADD   R2, Data(R1)
        LOAD  R1, Next(R1)
        JNNEG R1, Loop
Done    STORE R2, Sum
        SVC   SP, =HALT
```

Error, bug! Where?

Code Generation

- Last part of compilation
  - Can take 50% (or more) of compilation time
- Ordinary code generation
  - Initialization, expressions, control
- Generating optimized code
  - Code generation takes (much or very much) longer
  - **Execution happens (much or very much) faster**
  - When to keep the value of global/local variable X in a register and when not?
  - In which register the value of X should be kept?
    - In R1 always, in R5 during a loop, not in register

-- End --

- Electronic tube
  - Logic, memory
- ENIAC, 1945
  - Electronic Numerical Integrator and Computer
  - 1st general-purpose electronic digital computer
  - J.W. Mauchly, J.P. Eckert, J. von Neumann
  - 50m long
  - 17 468 electr. tubes
  - 5 000 additions/sec.
  - 357 mult./sec
  - Programming by rewiring