Corso di Architettura dei Sistemi a Microprocessore

Instruction Set Architecture

Luigi Coppolino

Contact info

Prof. Luigi Coppolino
luigi.coppolino@uniparthenope.it

Università degli Studi di Napoli "Parthenope"
Dipartimento di Ingegneria

Centro Direzionale di Napoli, Isola C4
V Piano lato SUD - Stanza n. 512

Tel: +39-081-5476702
Fax: +39-081-5476777

Roadmap

- Organizzazione della memoria
- Esempi di organizzazione della memoria
- I principali modi di indirizzamento
- Esempi d'impiego
- Riepilogo modi base
- Altri modi di indirizzamento
- Istruzioni ARM E COLDFIRE
- Uso dello Stack e chiamata di funzioni

Sources

- Textbook (chapter 2)
- Manuale Freescale (http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf)(http://www.freescale.com/files/dsp/doc/ref_manual/CFPRM.pdf)
- Manuale ARM (http://infocenter.arm.com/help/topic/com.arm.doc.dui0204j/DUI0204J_rvct_assembler_guide.pdf)
- Quick Guides:
  - ARM: http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001l/QRC0001_UAL.pdf
  - Coldfire (m68000): http://home.anadolu.edu.tr/~sgorgulu/micro2/2008/68KISx1.pdf

Address and Register size

- Address size:
  - Number of bits composing an address
  - If the Address size is *m* the address space of the memory it 2^m
- Register size:
  - Number of bits composing a register
- Typically Register Size is a multiple (or egual) of Memory Size

Register Transfer Notation

- Register transfer notation is used to describe hardware-level data transfers and operations
- Predefined names for procr. and I/O registers
- Arbitrary names for locations in memory
- Use […] to denote contents of a location
- Use ← to denote transfer to a destination
- Example: R2 ← [LOC]
  (transfer from LOC in memory to register R2)

Register Transfer Notation

- RTN can be extended to also show arithmetic operations involving locations
- Example: R4 ← [R2] + [R3]
  (add the contents of registers R2 and R3,
  place the sum in register R4)
- Right-hand expression always denotes a value, left-hand side always names a location

- Es.
  - : R4 ← [R2] + LOC

Assembly-Language Notation

- RTN shows data transfers and arithmetic
- Another notation needed to represent machine instructions & programs using them
- Assembly language is used for this purpose
- For the two preceding examples using RTN, the assembly-language instructions are:

```
Load    R2, LOC
Add     R4, R2, R3
```

Assembly-Language Notation

- An instruction specifies the desired operation and the operands that are involved
- We will use English words for the operations (e.g., Load, Store, and Add) when they are not related to a specific architecture
- Commercial processors use mnemonics, usually abbreviations (e.g., LD, ST, and ADD)
- Mnemonics differ from processor to processor
  - Will use mnemonics to report specific processors related code

Memory Operations

- Memory contains data & program instructions
- Control circuits initiate transfer of data and instructions between memory and processor
- *Read* operation: memory retrieves contents at address location given by processor
- *Write* operation: memory overwrites contents at given location with given data

Addressing Modes

- Programs use data structures to organize the information used in computations
- High-level languages enable programmers to describe operations for data structures
- Compiler translates into assembly language
- Addressing modes provide compiler with different ways to specify operand locations
- Consider modes used in RISC-style processors

RISC-type addressing modes.

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute	LOC	EA = LOC
Register indirect	(R*i*)	EA = [R*i*]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]

EA = effective address
Value = a signed number
X = index value

Addressing Modes

- We have already seen examples of the *register* and *absolute* addressing modes
- RISC-style instructions have a fixed size, hence absolute mode information limited to 16 bits
- Usually sign-extended to full 32-bit address Absolute mode is therefore limited to a subset of the full 32-bit address space
- Assume programs are limited to this subset

Variables

- Variable declaration in high-level language:
  IntegerNUM1, NUM2, SUM;
- Allocates storage to locations in the memory
- When referenced by high-level statements, compiler translates to assembly language:
  Load R2, NUM1
- Absolute mode (in subset of address space) enables access to variables in the memory

Constants

- Assume constant 200 is added to a variable
- *Immediate* mode enables use of constants in assembly-language instructions
- One approach for specification:

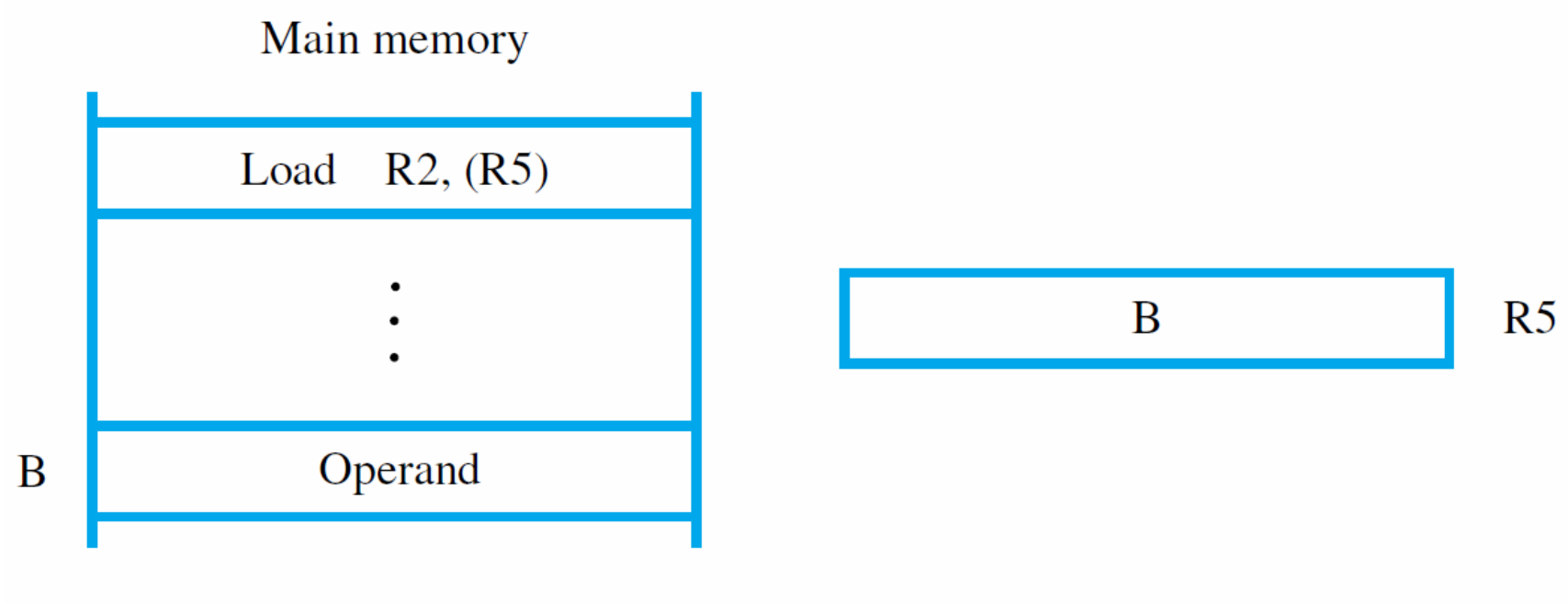
  Add R4, R6, $200_{immediate}$
- Not practical to use subscripts in this manner
- Alternative approach uses special character:

  Add R4, R6, #200

Indirection and Pointers

- Register, absolute, and immediate modes directly provide the operand or address
- Other modes provide information from which the effective address of operand is derived
- For program that adds numbers in a list, use register as pointer to next number
- *Indirect* mode provides address in register:

```
Load    R2, (R5)
```


Main memory
Load R2, (R5)
B
Operand
B
R5

Indirection and Pointers

- Body of loop can now use register as pointer
- To initialize the pointer, use the instruction:

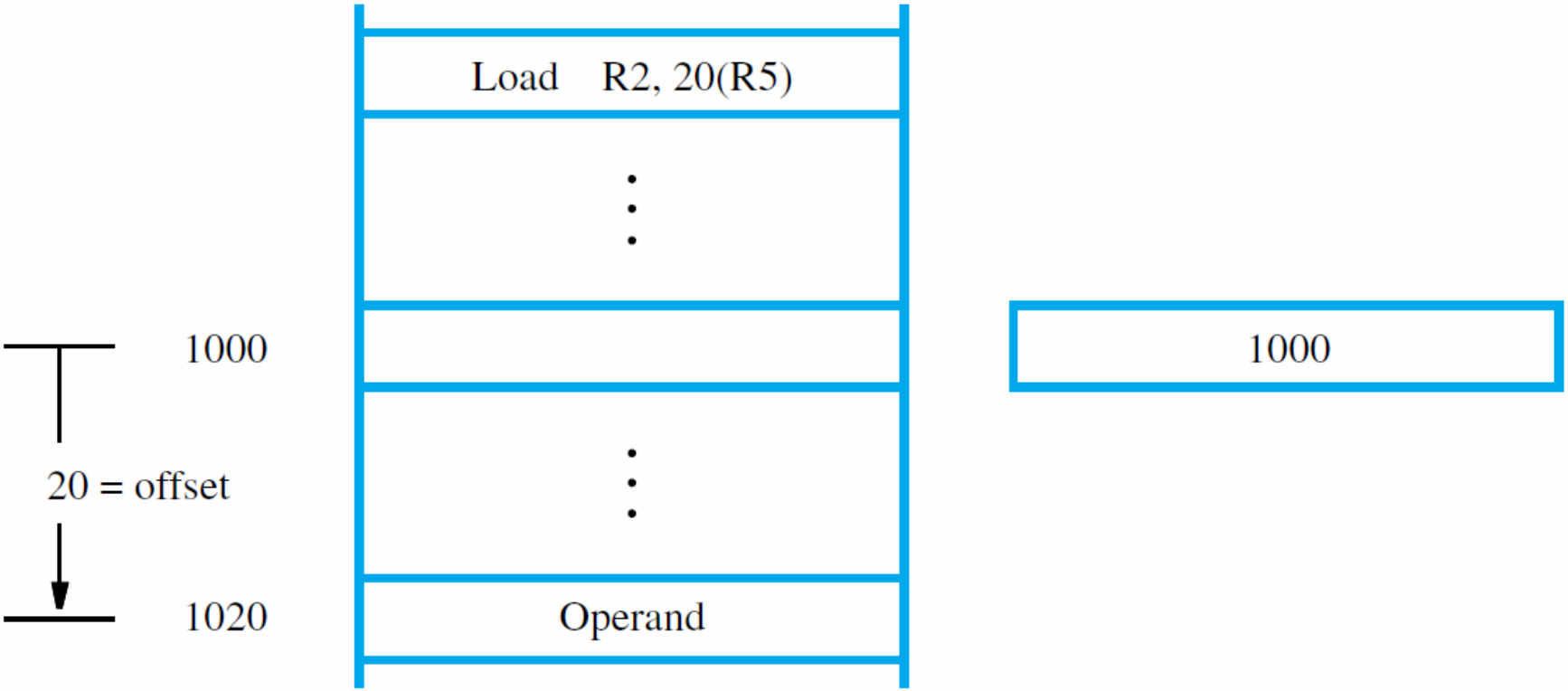
  Move R4, #NUM1
- In RISC-style processors, R0 is usually always 0
- Implement using Add and immediate mode:

  Add R4, R0, #NUM1
- Move is a convenient pseudoinstruction
- We now have complete list-addition program

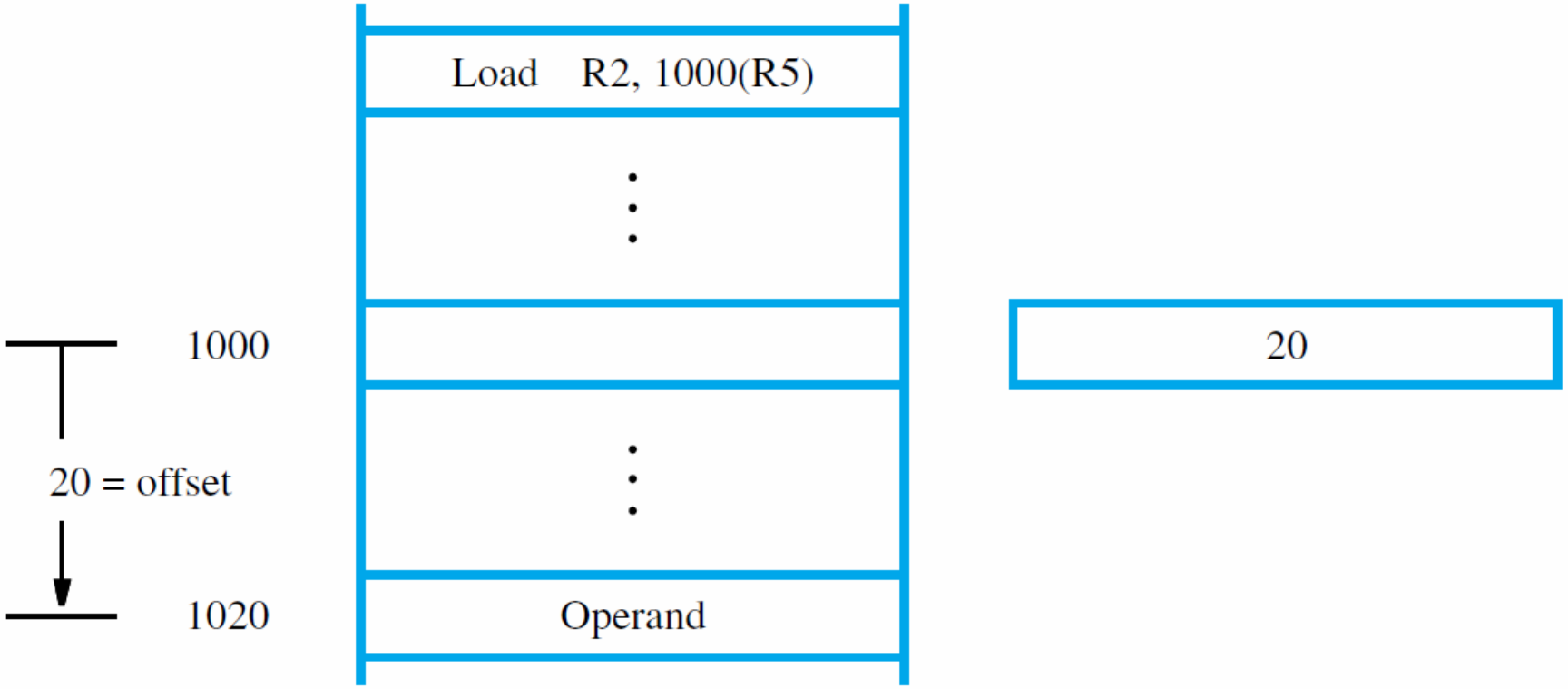
```
        Load                R2, N           Load the size of the list.
        Clear               R3              Initialize sum to 0.
        Move                R4, #NUM1       Get address of the first number.
LOOP:   Load                R5, (R4)        Get the next number.
        Add                 R3, R3, R5      Add this number to sum.
        Add                 R4, R4, #4      Increment the pointer to the list.
        Subtract            R2, R2, #1      Decrement the counter.
        Branch_if_[R2]>0    LOOP            Branch back if not finished.
        Store               R3, SUM         Store the final sum.
```

Indexing

- Consider *index* mode in: Load R2, *X*(R5)
- Effective address is given by [R5] + *X*
- For example, assume operand address is 1020, 4 words (20 bytes) from start of array at 1000
- Can put start address in R5 and use *X*=20
- Alternatively, put offset in R5 and use *X*=1000
- *Base with index* mode: Load R*k*, *X*(R*i*, R*j*)
- Effective address is given by [R*i*] + [R*j*] + *X*



(a) Offset is given as a constant



(b) Offset is in the index register

Additional Addressing Modes

- CISC style has other modes not usual for RISC
- *Autoincrement* mode: effective address given by register contents; after accessing operand, register contents incremented to point to next
- Useful for adjusting pointers in loop body:

  Add SUM, (R*i*)+

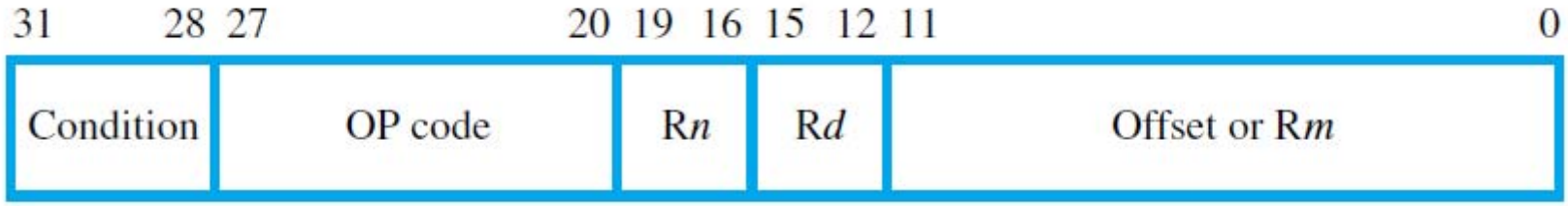
  MoveByte (R*j*)+, R*k*
- Increment by 4 for words, and by 1 for bytes

Arm addressing modes

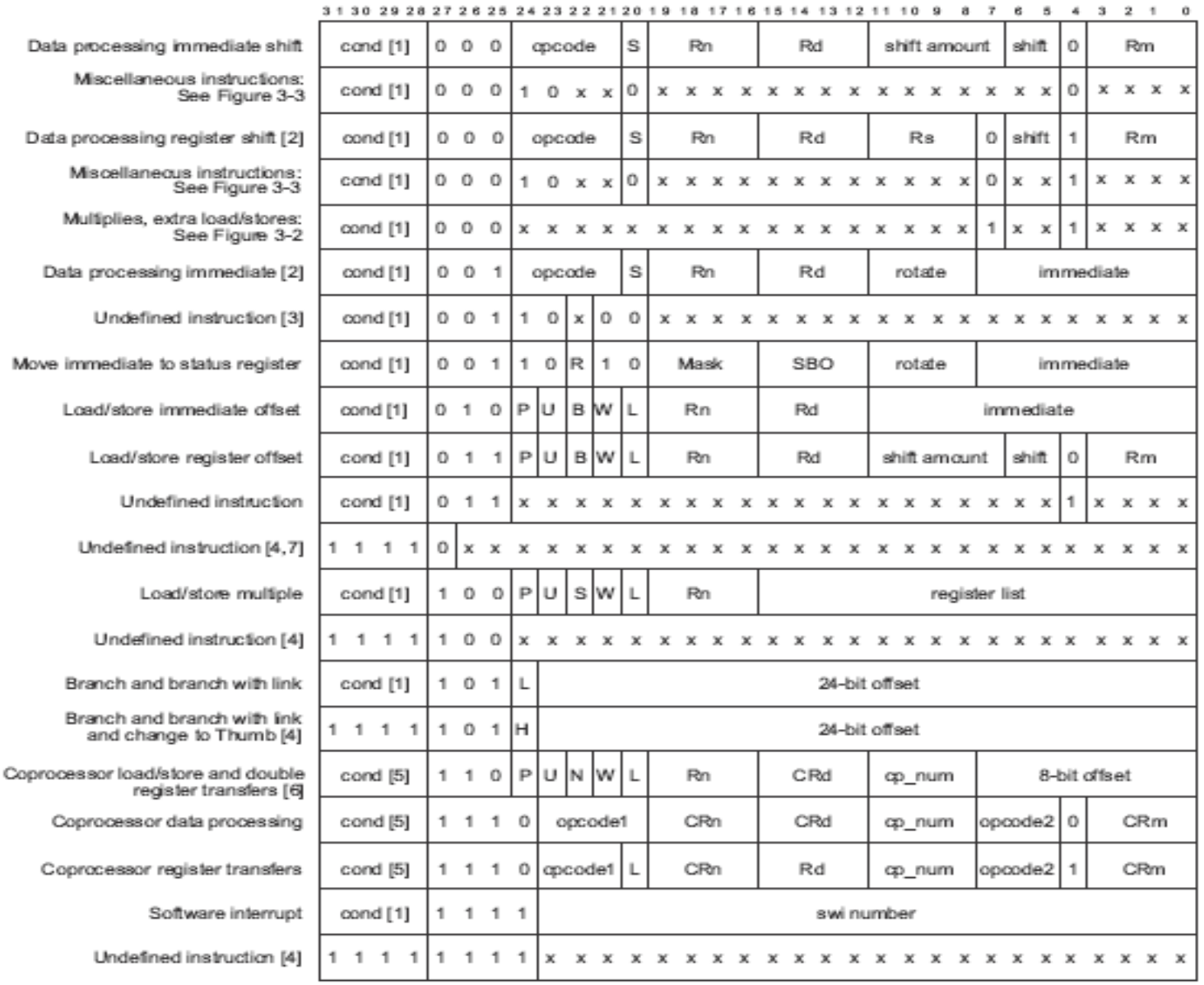
Addressing modes

- All modes are derived from a basic form of indexed addressing
- The effective address of a memory operand is the sum of the contents of a base register R*n* and a signed offset
- The offset is either a 12-bit immediate value in the instruction or the contents of a second register R*m*
- Examples of addressing modes can be shown by using the Load instruction LDR, whose format is given in following slide
- The store instruction STR has same format
- Both LDR and STR access a word location

LOAD Instruction Encoding



Instruction Structure

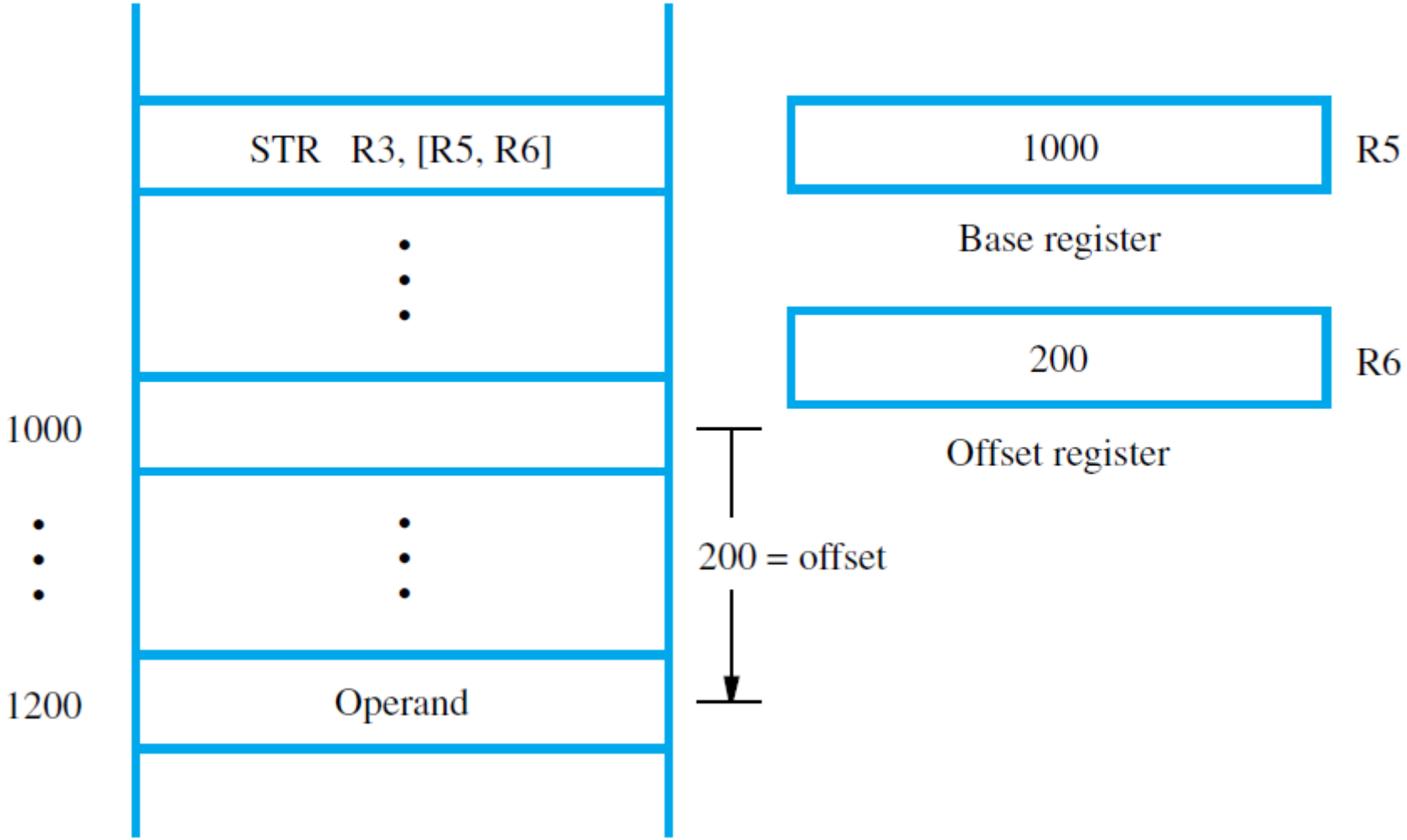


Addressing modes

- Pre-indexed mode:

LDR R*d*, [R*n*, #offset]
performs
R*d* ← [[R*n*] + offset]

LDR R*d*, [R*n*, R*m*]
performs
R*d* ← [[R*n*] + [R*m*]]


STR R3, [R5, R6]
1000
1200
Operand
200 = offset
1000
R5
Base register
200
R6
Offset register

Addressing modes

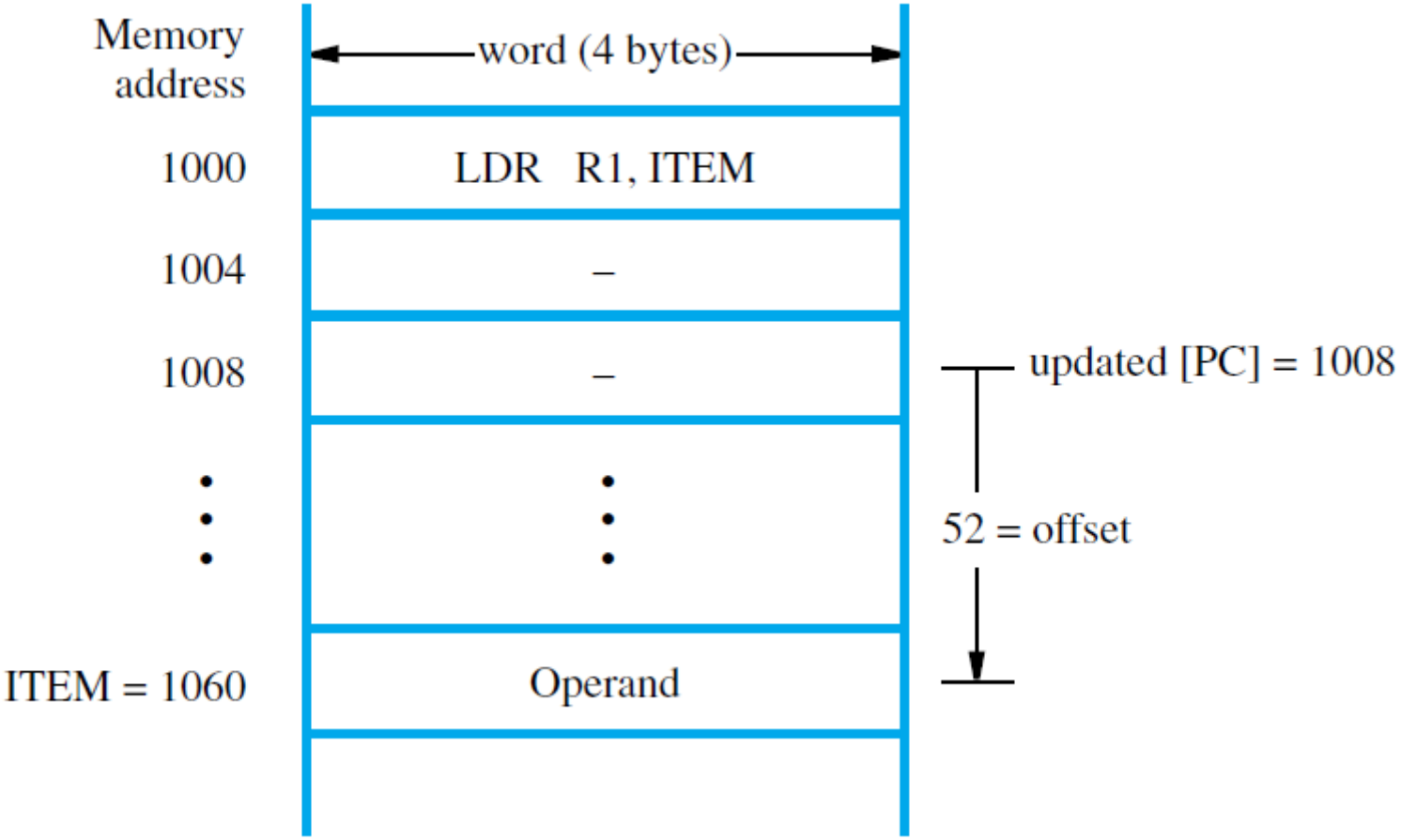
- Relative mode:

  LDR R*d*, ITEM

  performs

  R*d* ← [[PC] + offset]

  where offset is calculated by the assembler


Memory address
word (4 bytes)
1000
LDR R1, ITEM
1004
–
1008
–
updated [PC] = 1008
52 = offset
ITEM = 1060
Operand

Addressing modes

- Pre-indexed with writeback (a generalization of the autodecrement mode):

  LDR R*d*, [R*n*, #offset]!

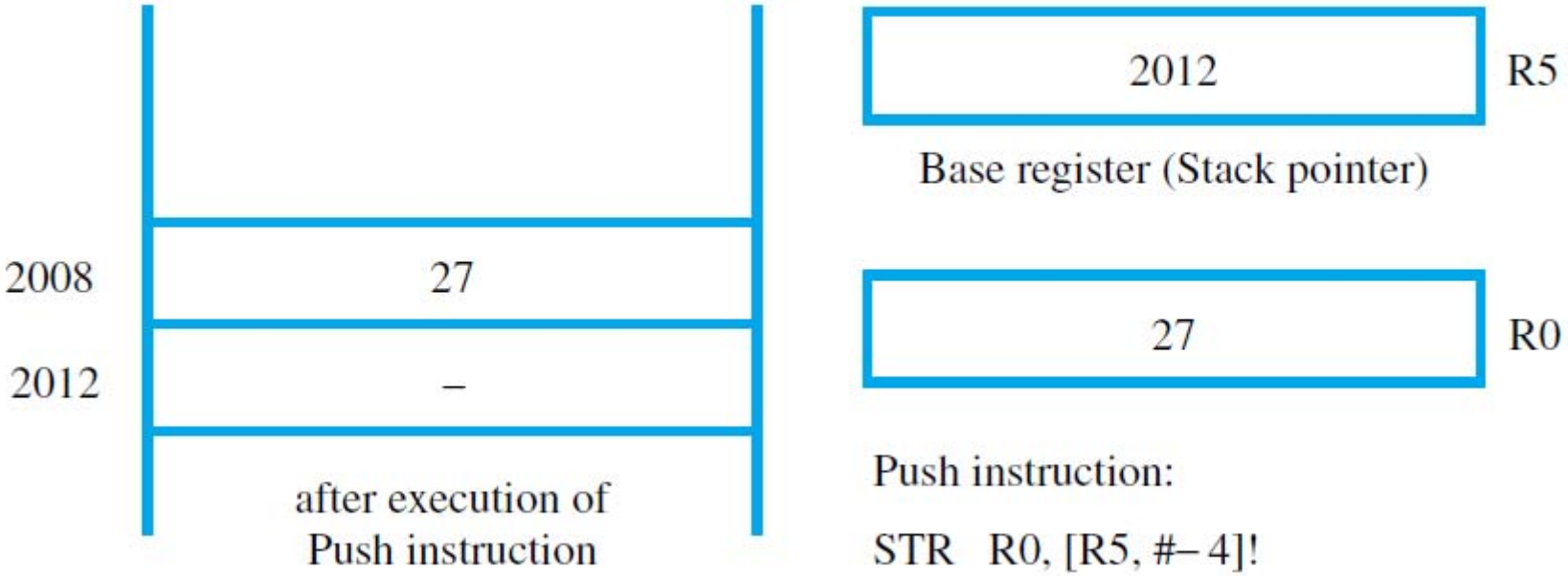
  performs

  R*d* ← [[R*n*] + offset]

  followed by

  R*n* ← [R*n*] + offset

  (R*m* can be used instead of #offset)


2008
27
2012
–
after execution of
Push instruction
2012
R5
Base register (Stack pointer)
27
R0
Push instruction:
STR R0, [R5, #– 4]!

Addressing modes

- Post-indexed mode (a generalization of the autoincrement mode):

  LDR R*d*, [R*n*], #offset

  performs

  R*d* ← [[R*n*]]

  followed by

  R*n* ← [R*n*] + offset

  (R*m* can be used instead of #offset)

Addressing modes

- If the offset is given as the contents of R*m*,
  it can be shifted before being used

  Example:

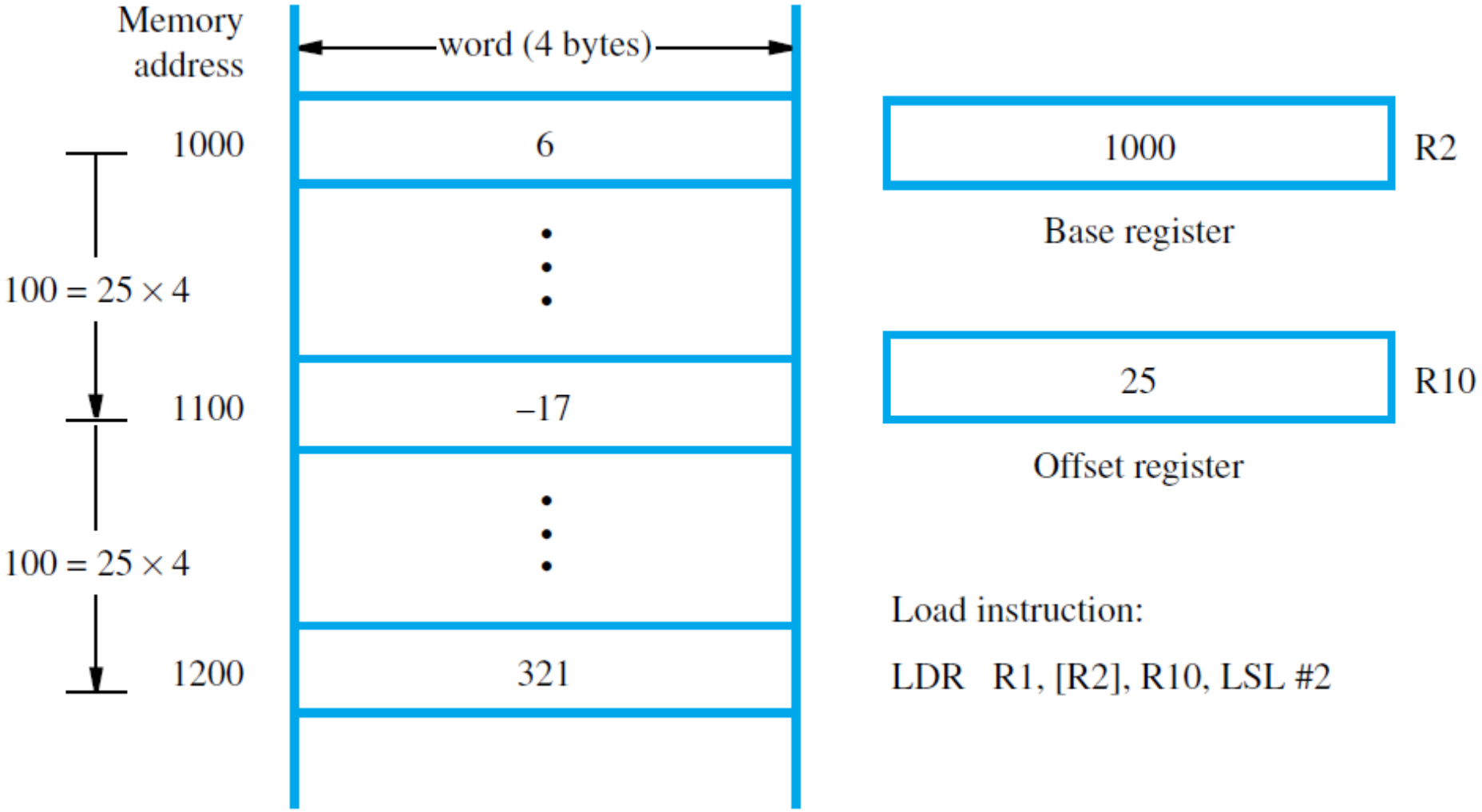
  LDR R0, [R1, –R2, LSL #4]!

  performs

  R0 ←[[R1] – 16 × [R2]]

  followed by

  R1 ← [R1] – 16 × [R2]


Memory address
word (4 bytes)
1000
6
100 = 25 × 4
1100
–17
100 = 25 × 4
1200
321
1000
R2
Base register
25
R10
Offset register
Load instruction:
LDR R1, [R2], R10, LSL #2

ARM indexed addressing modes.

Name	Assembler syntax	Addressing function
With immediate offset:		
Pre-indexed	[R*n*, #offset]	EA = [R*n*] + offset
Pre-indexed with writeback	[R*n*, #offset]!	EA = [R*n*] + offset; R*n* ← [R*n*] + offset
Post-indexed	[R*n*], #offset	EA = [R*n*]; R*n* ← [R*n*] + offset
With offset magnitude in R*m*:		
Pre-indexed	[R*n*, ± R*m*, shift]	EA = [R*n*] ± [R*m*] shifted
Pre-indexed with writeback	[R*n*, ± R*m*, shift]!	EA = [R*n*] ± [R*m*] shifted; R*n* ← [R*n*] ± [R*m*] shifted
Post-indexed	[R*n*], ± R*m*, shift	EA = [R*n*]; R*n* ← [R*n*] ± [R*m*] shifted
Relative (Pre-indexed with immediate offset)	Location	EA = Location = [PC] + offset

EA = effective address
offset = a signed number contained in the instruction
shift = direction #integer
where direction is LSL for left shift or LSR for right shift; and
integer is a 5-bit unsigned number specifying the shift amount
±R*m* = the offset magnitude in register R*m* can be added to or subtracted from the contents of base register R*n*

ColdFire addressing modes

Instructions

- One, two, or three consecutive words
- *OP-code* word is first – it specifies operation
- Also provides some addressing information; one or two *extension* words provide more
- Most arithmetic and data-transfer instructions have source/destination operands:

  OP src, dst
- .L, W., or .B suffix for OP code specifies size