Thực hành
Kiến trúc máy tính

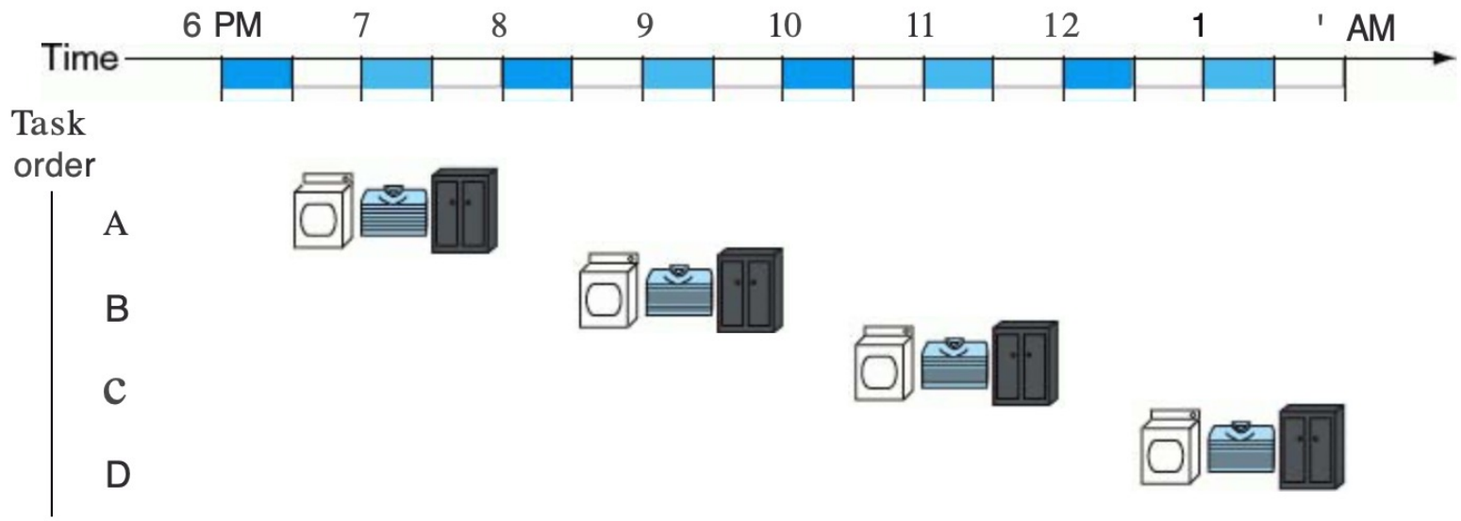
Giảng viên: Nguyễn Thị Thanh Nga
Khoa Kỹ thuật máy tính
Trường CNTT&TT

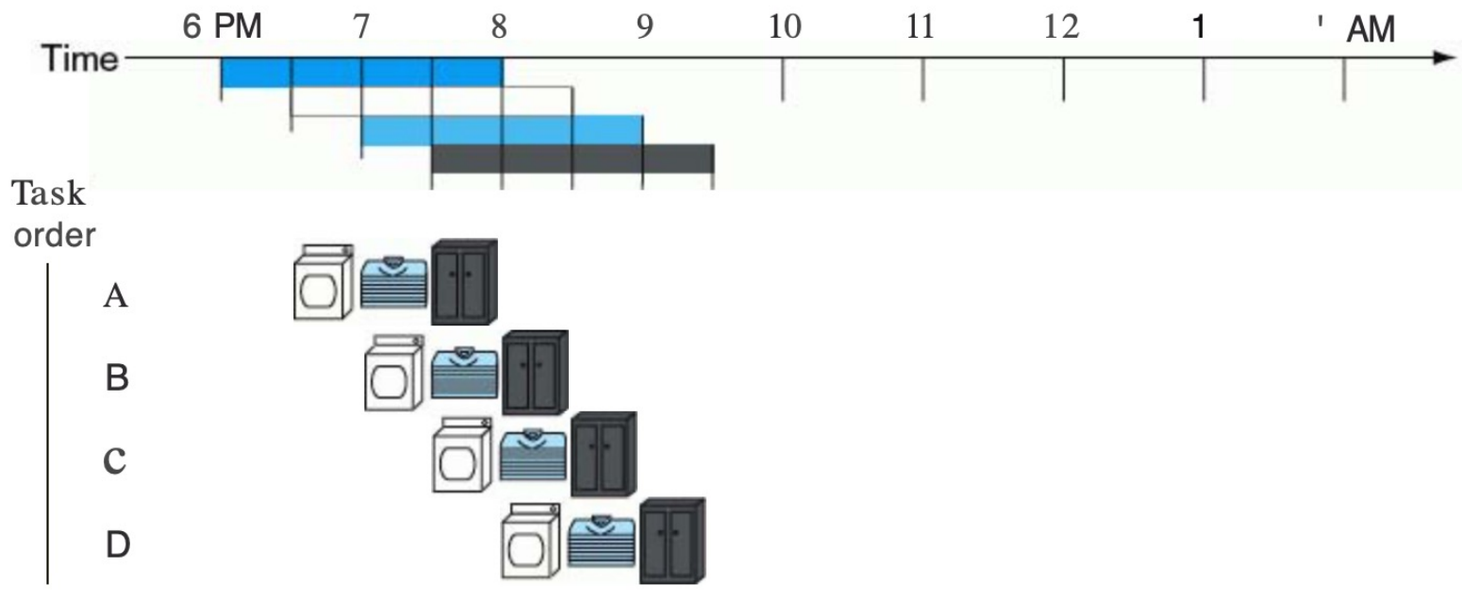
Tuần 7

- Kiến trúc Pipeline
- Giới thiệu các dịch vụ hệ thống (Syscall)
- Một số dịch vụ hệ thống thường dùng
  - 1, 4, 5, 8, 10, 11, 12, 17, 31, 33, 50, 54, 55, 56, 59

Kỹ thuật Pipeline

- Là kỹ thuật trong đó các lệnh được thực thi theo kiểu chồng lấn lên nhau (overlap)





Kỹ thuật Pipeline

- Pipeline được chia thành các giai đoạn và các giai đoạn này được kết nối với nhau để tạo thành một cấu trúc giống như một đường ống.

- Các lệnh sẽ được đưa vào một đầu và ra từ đầu còn lại.

- Pipeline làm tăng thông lượng lệnh, có nghĩa là tiêu tốn ít thời gian hơn để hoàn tất cùng một lượng công việc so với tuần tự.

- Chú ý: pipeline không làm giảm thời gian hoàn thành 1 công việc mà làm giảm thời gian hoàn thành tổng số công việc.

Kỹ thuật Pipeline

Khi thực thi, các lệnh được chia thành 5 giai đoạn:

- **Giai đoạn 1: Nạp lệnh**
- **Giai đoạn 2: Giải mã lệnh**
- **Giai đoạn 3: Thực thi lệnh**
- **Giai đoạn 4: Truy cập bộ nhớ**
- **Giai đoạn 5: Ghi kết quả**

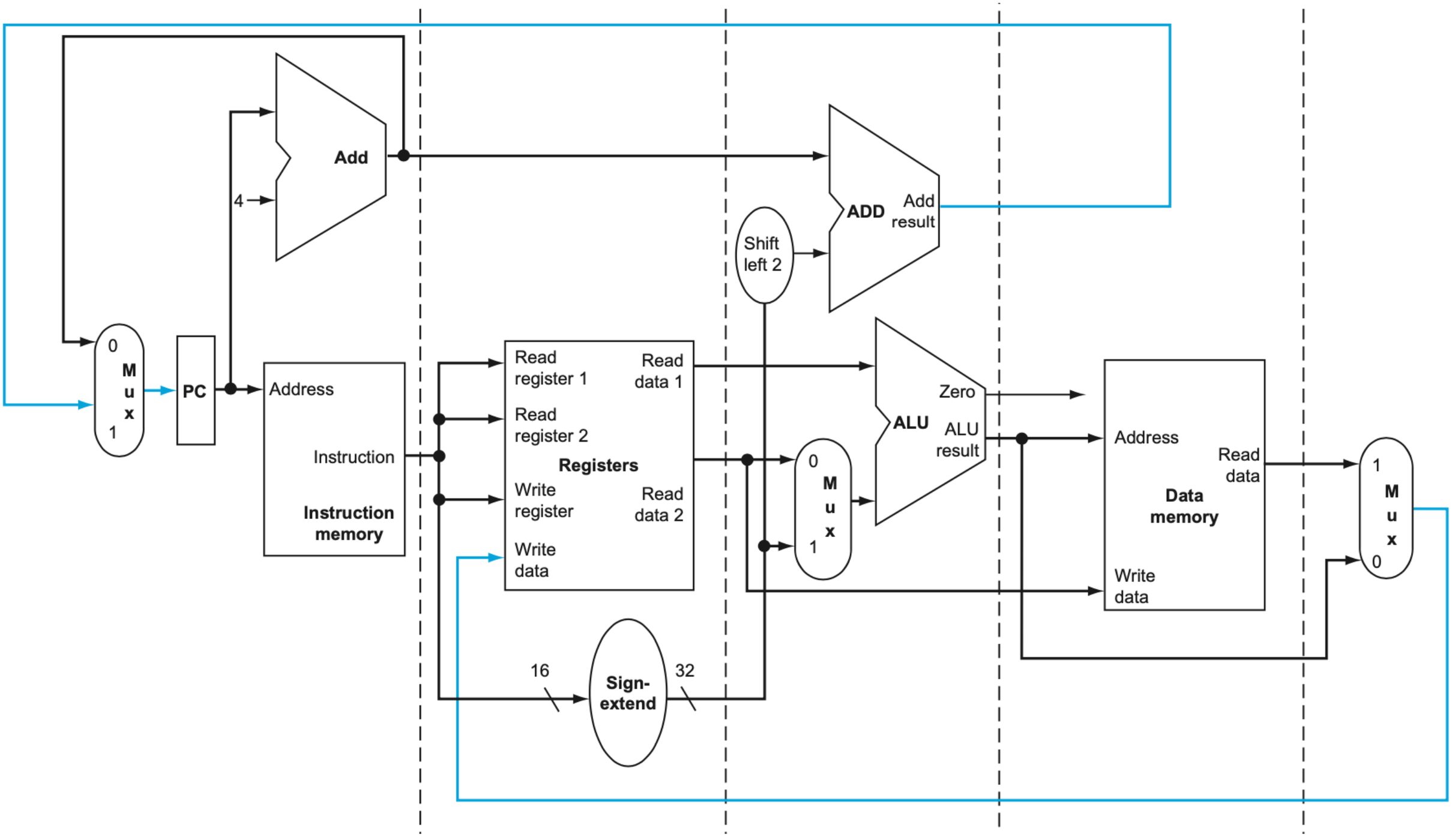
1. Nạp lệnh (IF)

2. Giải mã lệnh (ID)

3. Thực thi (EX)

4. Truy cập bộ nhớ (MEM)

5. Ghi kết quả (WB)



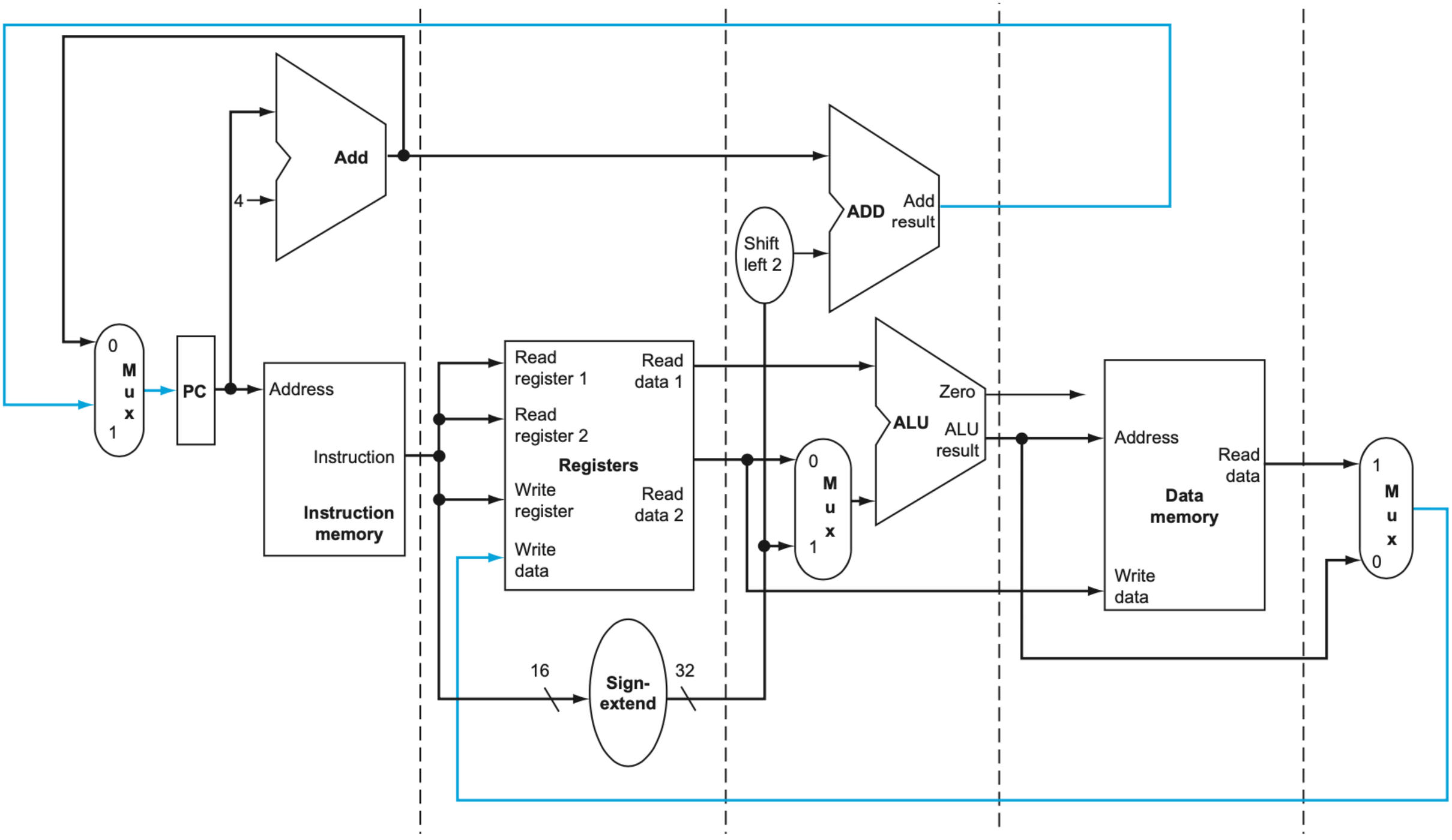
1. Nạp lệnh (IF)

2. Giải mã lệnh (ID)

3. Thực thi (EX)

4. Truy cập bộ nhớ (MEM)

5. Ghi kết quả (WB)



Giai đoạn 1: CPU đọc lệnh từ địa chỉ bộ nhớ mà giá trị được lưu trong bộ đếm chương trình $pc

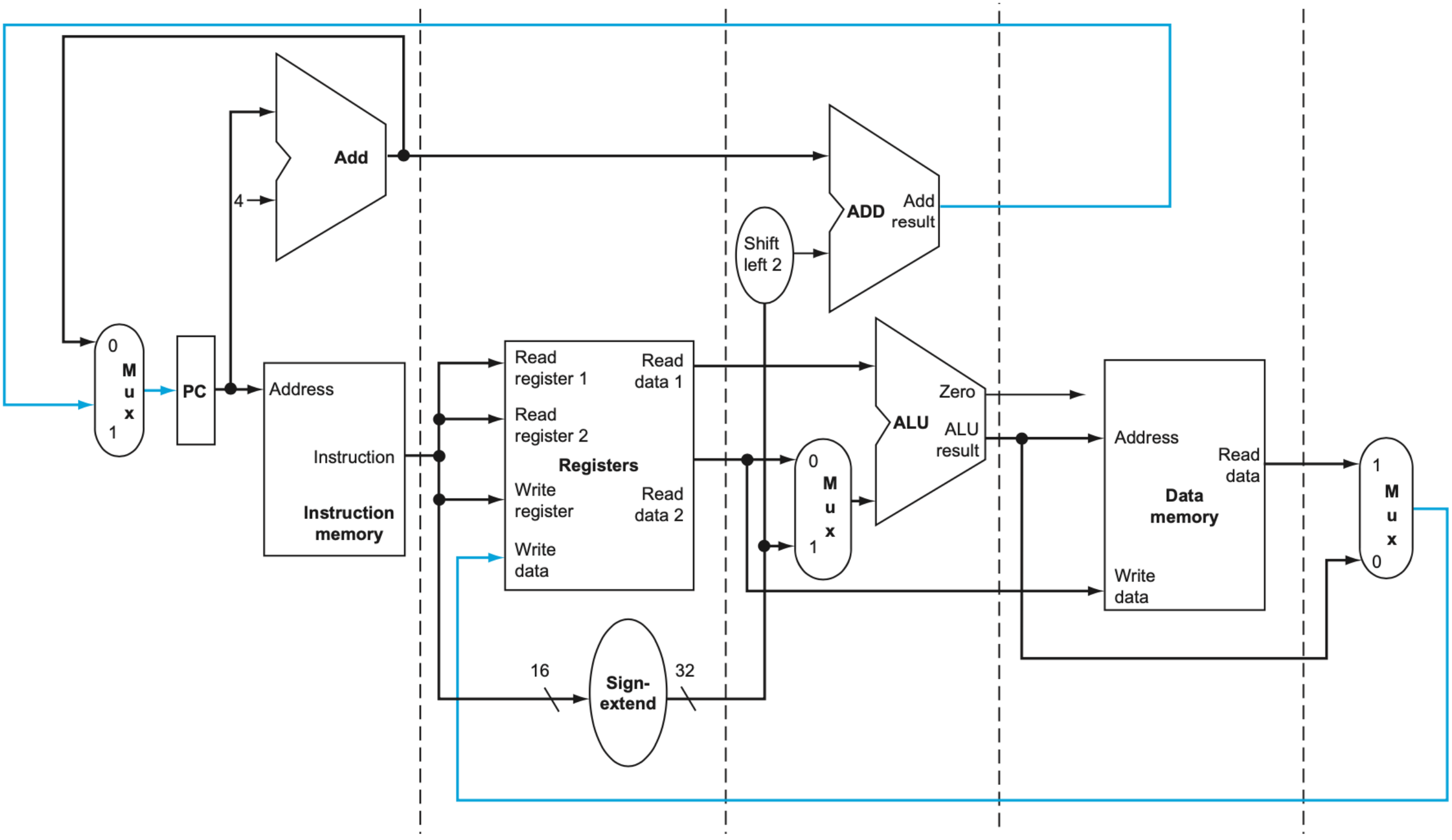
1. Nạp lệnh (IF)

2. Giải mã lệnh (ID)

3. Thực thi (EX)

4. Truy cập bộ nhớ (MEM)

5. Ghi kết quả (WB)



Giai đoạn 2: lệnh được giải mã và lấy các giá trị lưu trong thanh ghi để chuẩn bị thực hiện lệnh

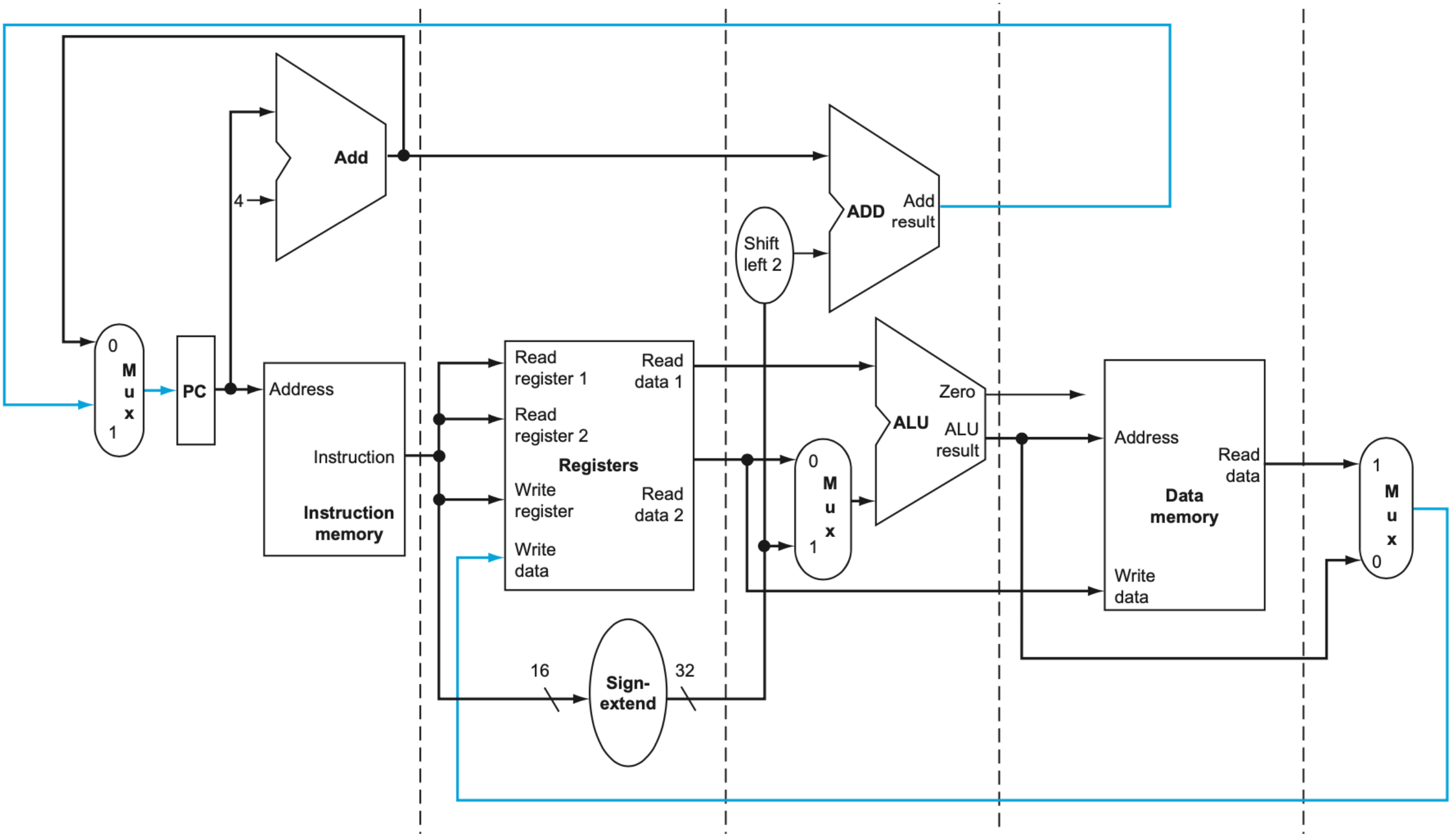
1. Nạp lệnh (IF)

2. Giải mã lệnh (ID)

3. Thực thi (EX)

4. Truy cập bộ nhớ (MEM)

5. Ghi kết quả (WB)



Giai đoạn 3: ALU thực thi lệnh.

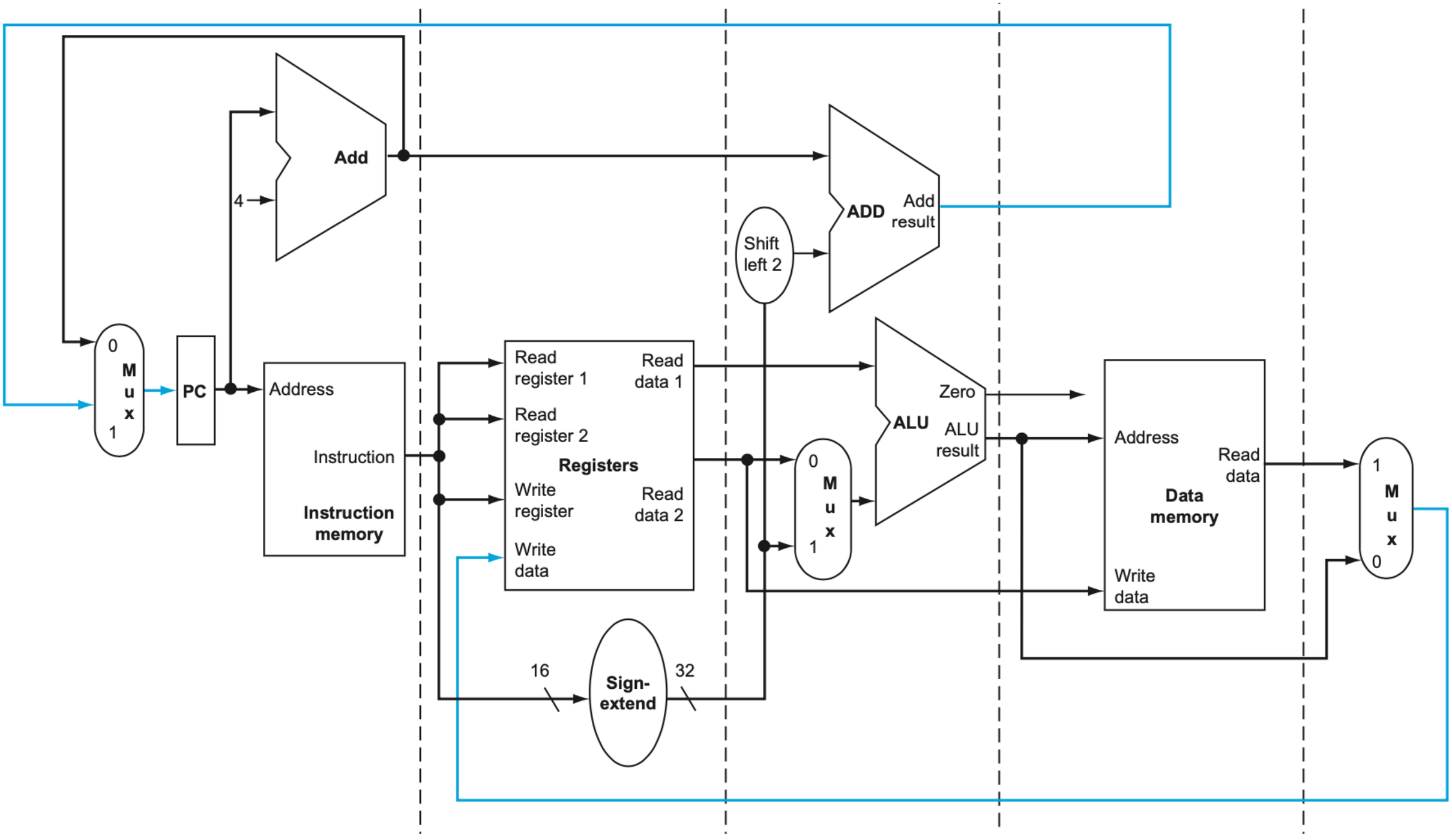
1. Nạp lệnh (IF)

2. Giải mã lệnh (ID)

3. Thực thi (EX)

4. Truy cập bộ nhớ (MEM)

5. Ghi kết quả (WB)



Giai đoạn 4: Toán hạng bộ nhớ được đọc và ghi từ/đến bộ nhớ trong lệnh.

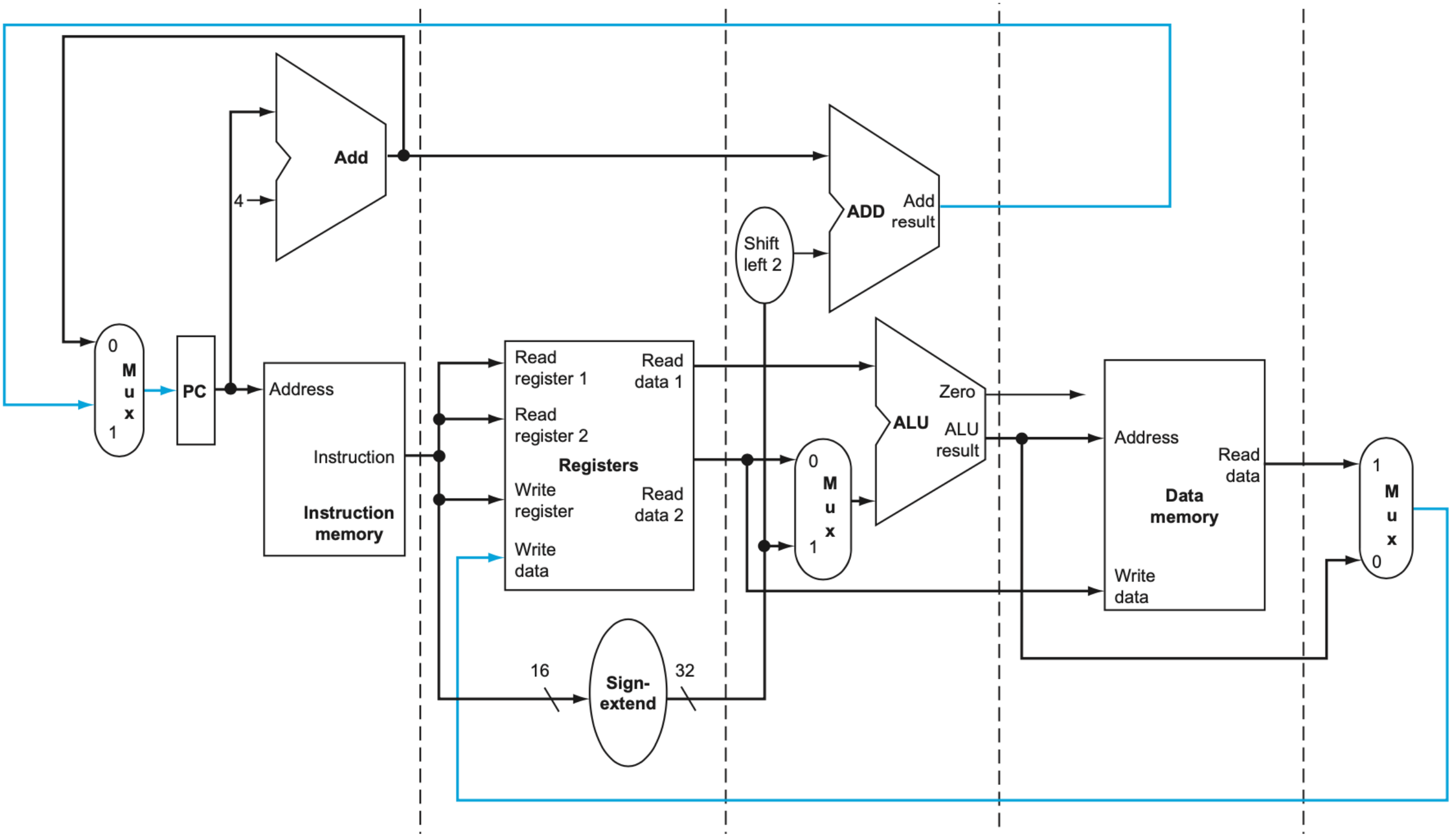
1. Nạp lệnh (IF)

2. Giải mã lệnh (ID)

3. Thực thi (EX)

4. Truy cập bộ nhớ (MEM)

5. Ghi kết quả (WB)



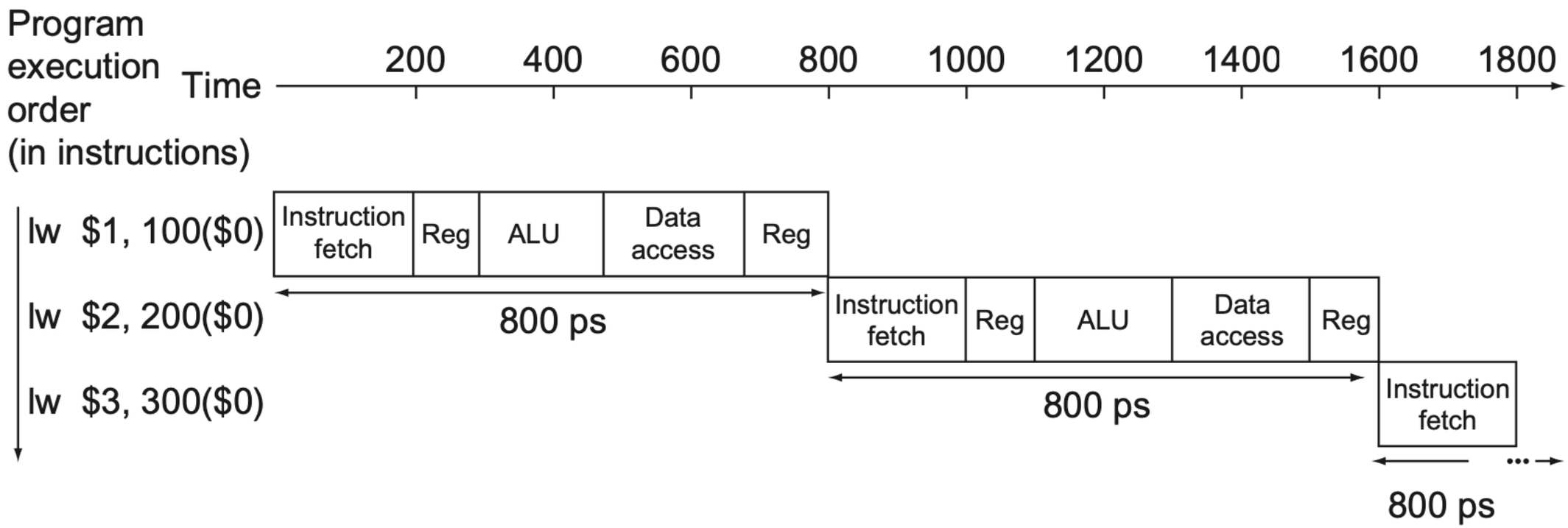
Giai đoạn 5: các giá trị được tính toán/nạp được ghi trở lại vào thanh ghi có trong lệnh.

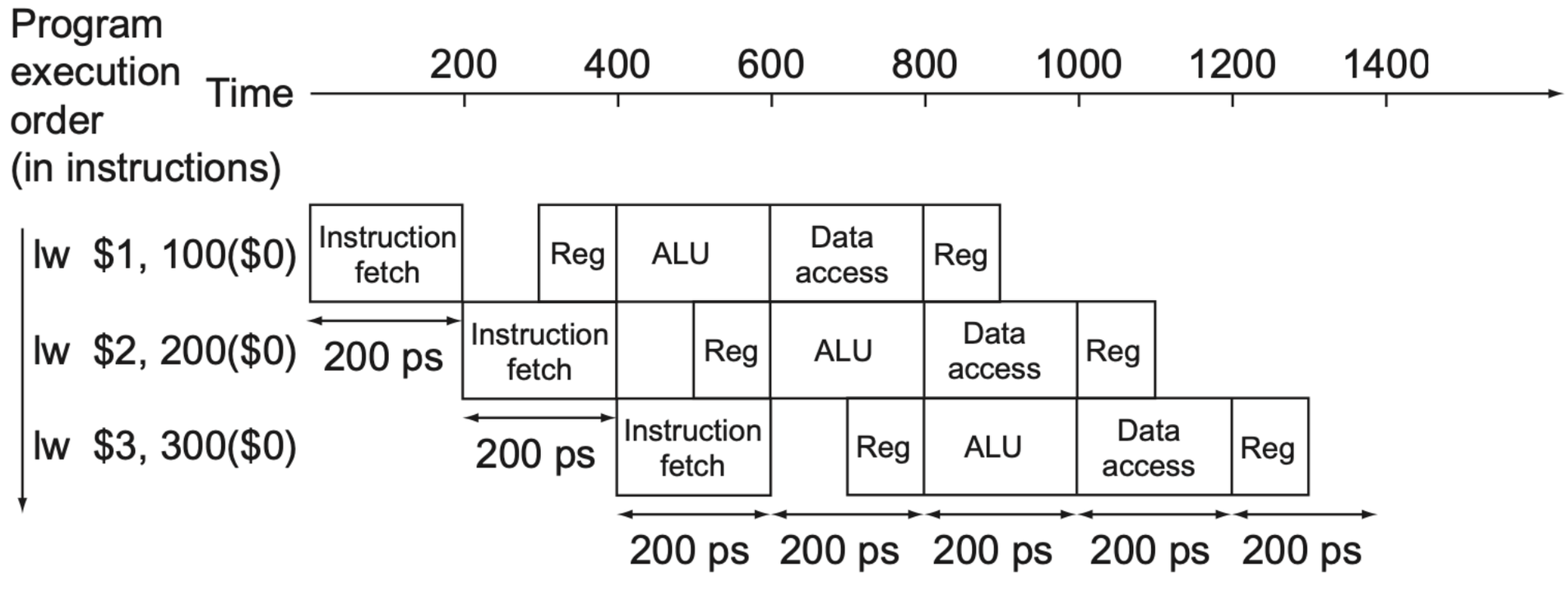
Kỹ thuật Pipeline

- Xét bộ xử lý với 8 lệnh cơ bản, tương ứng với thời gian hoạt động cho từng giai đoạn như sau:

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

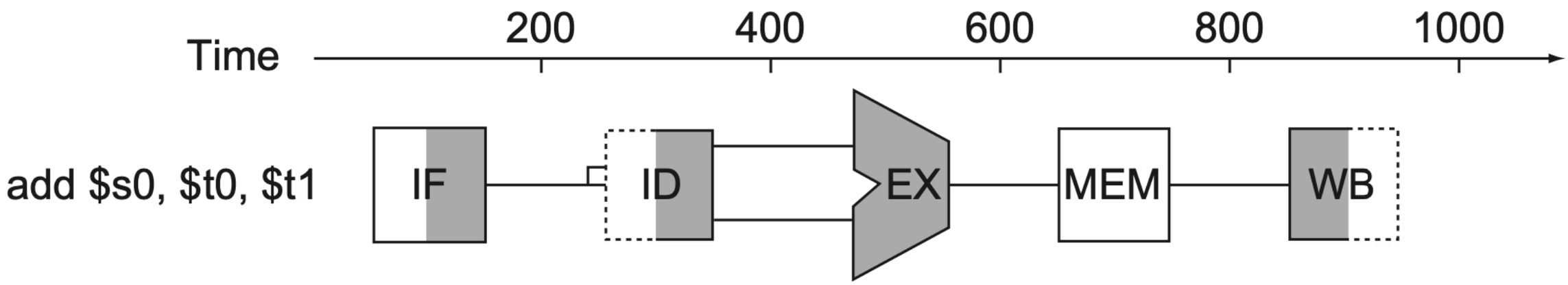
Kỹ thuật Pipeline





Kỹ thuật Pipeline

- 5 giai đoạn thực thi một lệnh:



- Lện không làm gì được tô trắng và ngược lại.
- Nửa phải tô đen là thao tác đọc, nửa trái tô đen là thao tác ghi.

Xung đột

- Các xung đột có thể xảy ra khi áp dụng kỹ thuật pipeline.

- Xung đột là trạng thái mà lệnh tiếp theo không thể thực thi trong chu kỳ pipeline ngay sau đó (hoặc thực thi nhưng sẽ cho kết quả sai), thường do một trong ba nguyên nhân sau:

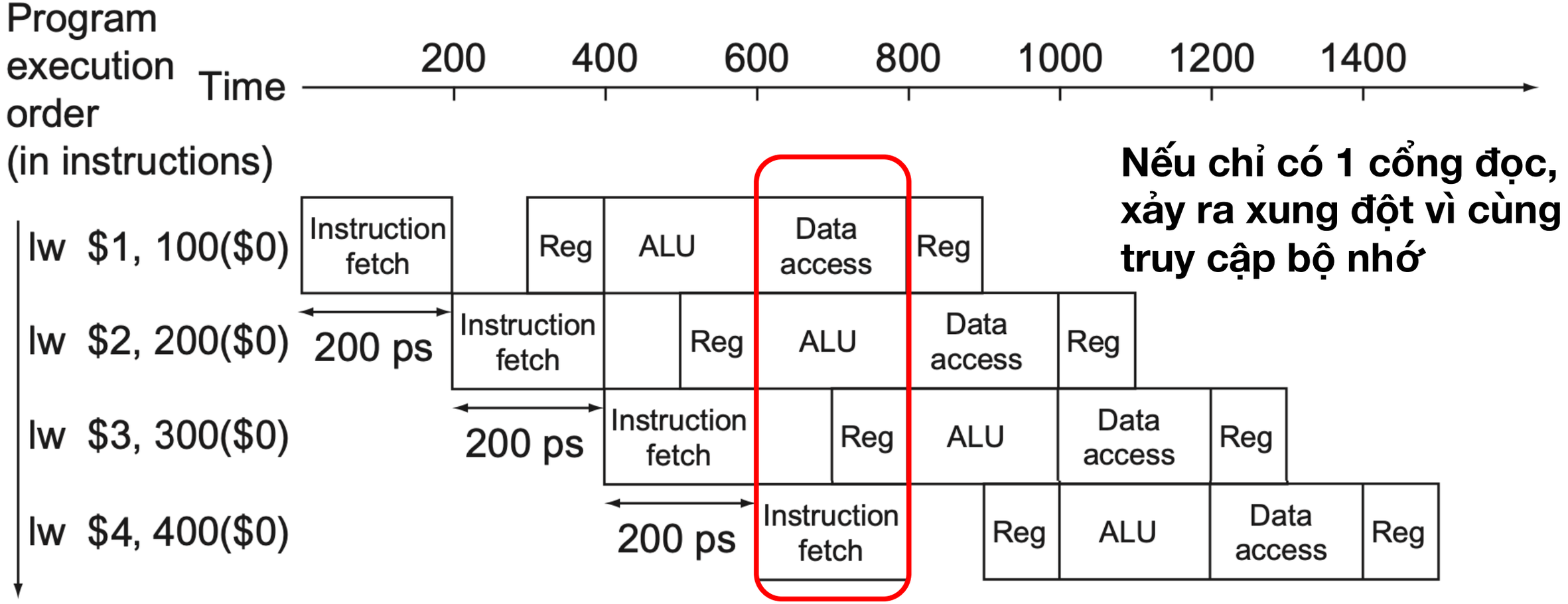
  - Xung đột cấu trúc

  - Xung đột dữ liệu

  - Xung đột điều khiển

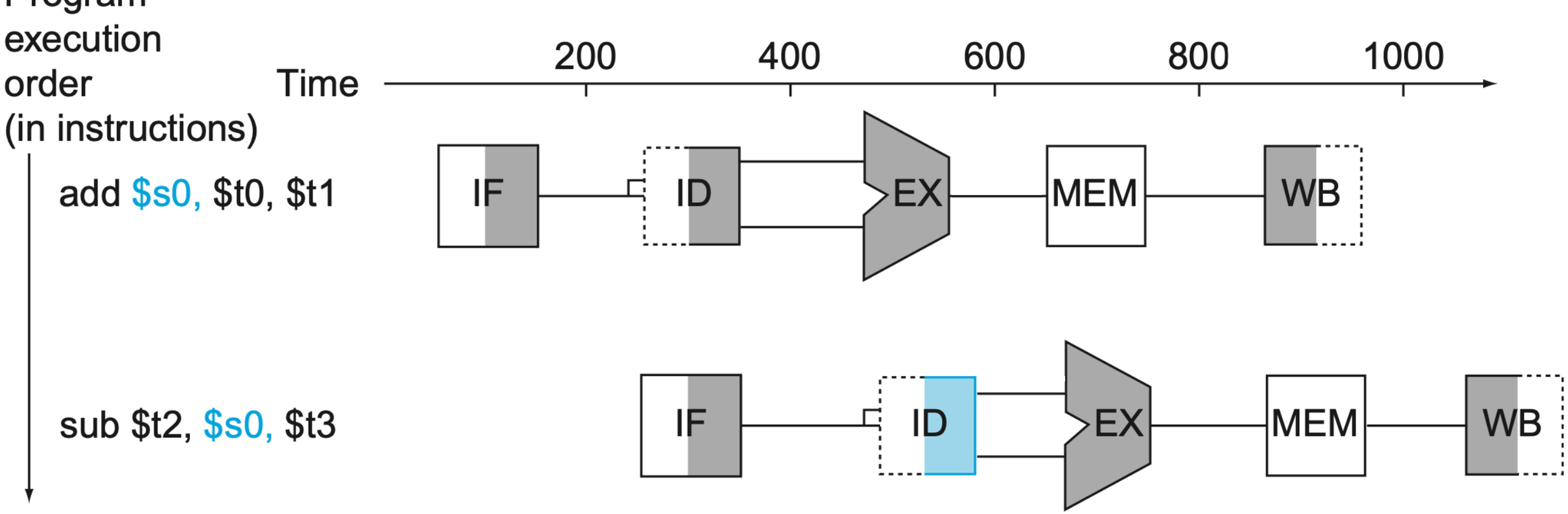
Xung đột cấu trúc

- Khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do phần cứng không thể hỗ trợ (đồng thời cùng truy xuất vào một tài nguyên phần cứng)



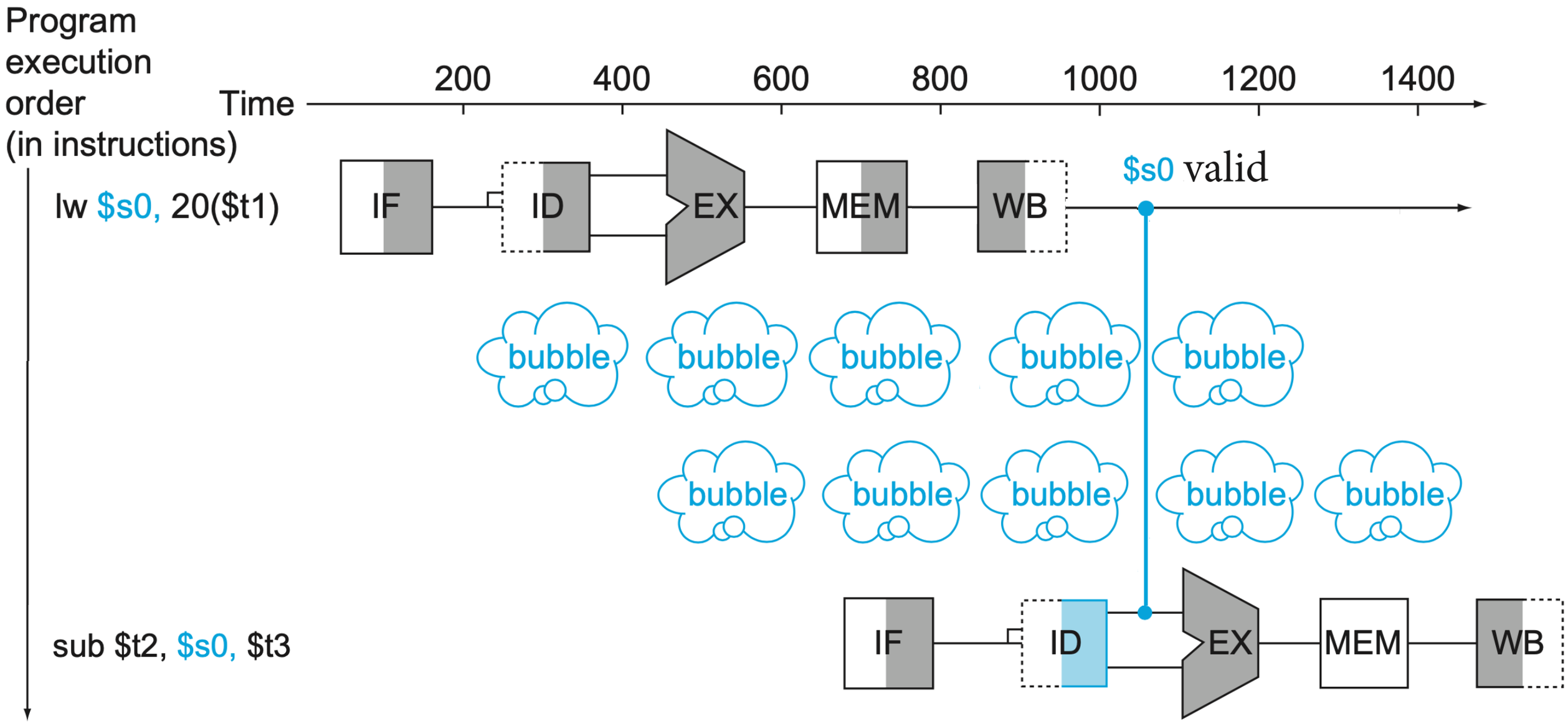
Xung đột dữ liệu

- Khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do dữ liệu mà lệnh này cần vẫn chưa sẵn sàng.



Xung đột dữ liệu

- Chờ thêm 2 chu kỳ xung clock :



Xung đột điều khiển

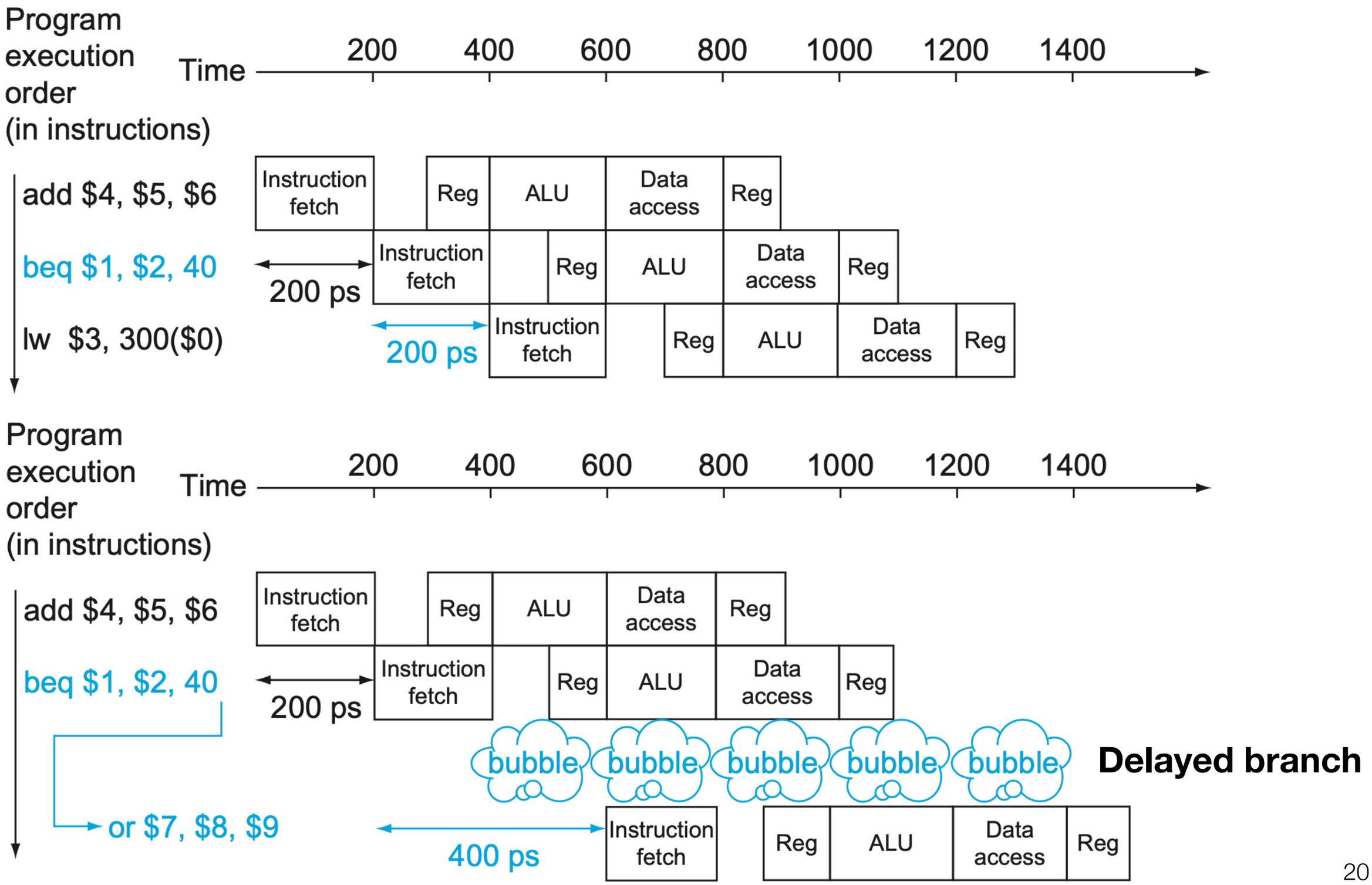
- Khi pipeline dự đoán rẽ nhanh sai và do đó nạp các lệnh vào pineline thì sau đó các lệnh này phải bị loại bỏ.

- Khái niệm xung đột rẽ nhánh cũng đề cập đến xung đột điều khiển.

- Xét đoạn chương trình sau:

```
add $4, $5, $6

beq $1, $2, label

lw $3, 300($0)
```

Xung đột điều khiển



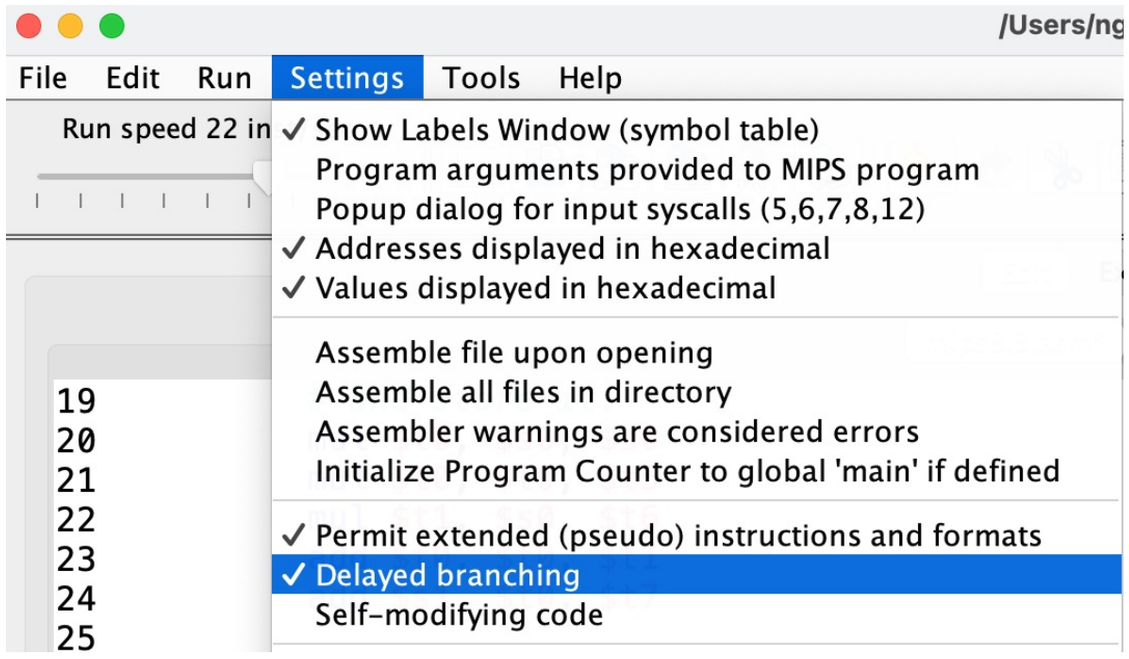
Bài tập 7.1

- Nhập vào chương trình sau:

```
.text
.globl main
        # Get input value and store it in $s0
main:   la $a0, prompt
        jal PromptInt
        move $s0, $v0

        # Load constants a, b, and c into registers
        li $t5, 5
        li $t6, 2
        li $t7, 3

        # Calculate the result of
        #y=a*x*x + b * x + c and store it
        mul $t0, $s0, $s0
        mul $t0, $t0, $t5
        mul $t1, $s0, $t6
        add $t0, $t0, $t1
        add $s1, $t0, $t7

        # Print output from memory y
        la $a0, result
        move $a1, $s1
        jal PrintInt
        jal PrintNewLine

        #Exit program
        jal Exit
.data
   y: .word 0
   prompt: .asciiz "Enter a value for x: "
   result: .asciiz "The result is: "

.include "utils.asm"
```

Bài tập 7.1

- Biên dịch và chạy chương trình
- Chọn **Delayed branching** trong **Setting**



- Biên dịch và chạy chương trình một lần nữa
- Giải thích kết quả và giải quyết vấn đề.

SYSCALL

- Các dịch vụ hệ thống, chủ yếu dành cho vào và ra, sử dụng trong các chương trình MIPS.
- Nội dung các thanh ghi MIPS không bị ảnh hưởng bởi lời gọi hệ thống, trừ các thanh ghi kết quả.

Cách sử dụng các dịch vụ SYSCALL

- Nạp số dịch vụ vào thanh ghi **$v0**.
- Nạp giá trị tham số, nếu có, vào **$a0**, **$a1**, **$a2**, hay **$a3** như đã chỉ định.
- Gọi lệnh SYSCALL.
- Các giá trị trả về nếu có, lấy từ các thanh ghi đã được chỉ định trước.

Bảng các dịch vụ khả dụng thường dùng

- Tham chiếu đến bảng sau: Table of available services.

- http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

1. In số nguyên hệ 10

- In một số nguyên thập phân ra đầu ra chuẩn (màn hình)

Argument(s):

$v0 = 1
$a0 = number to be printed

Return value:

none

Example:

```
li $v0, 1       # service 1 is print integer
li $a0, 0x307   # the interger to be printed is 0x307
syscall         # execute
```

1. In số nguyên hệ 10

- In một số nguyên thập phân ra đầu ra chuẩn (màn hình)

Argument(s):

$v0 = 1
$a0 = number to be printed

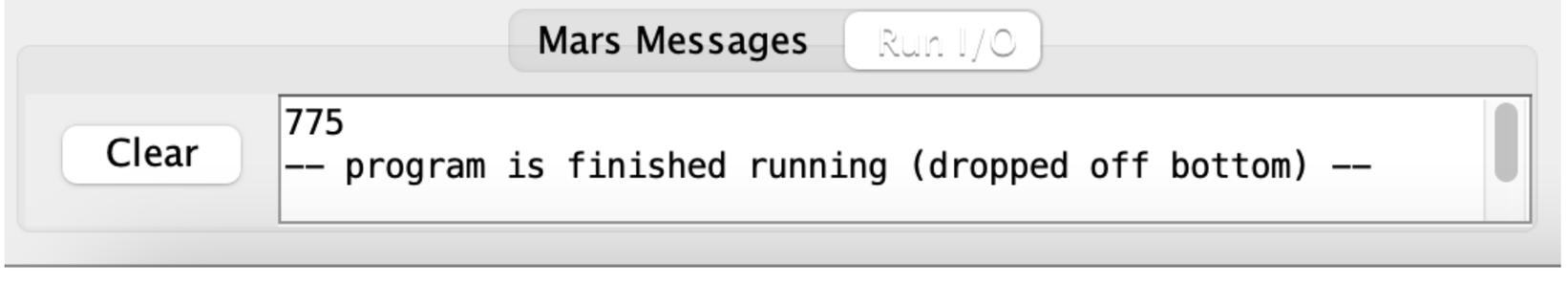
Return value:

none

Example:

```
li $v0, 1       # service 1 is print integer
li $a0, 0x307   # the interger to be printed is 0x307
syscall         # execute
```

- Kết quả



2. MessageDialogInt

- Hiển thị một số nguyên dưới dạng hộp thoại:

Argument(s):

$v0 = 56
$a0 = address of the null-terminated message string
$a1 = int value to display in string form after the first string *Return value:*

none

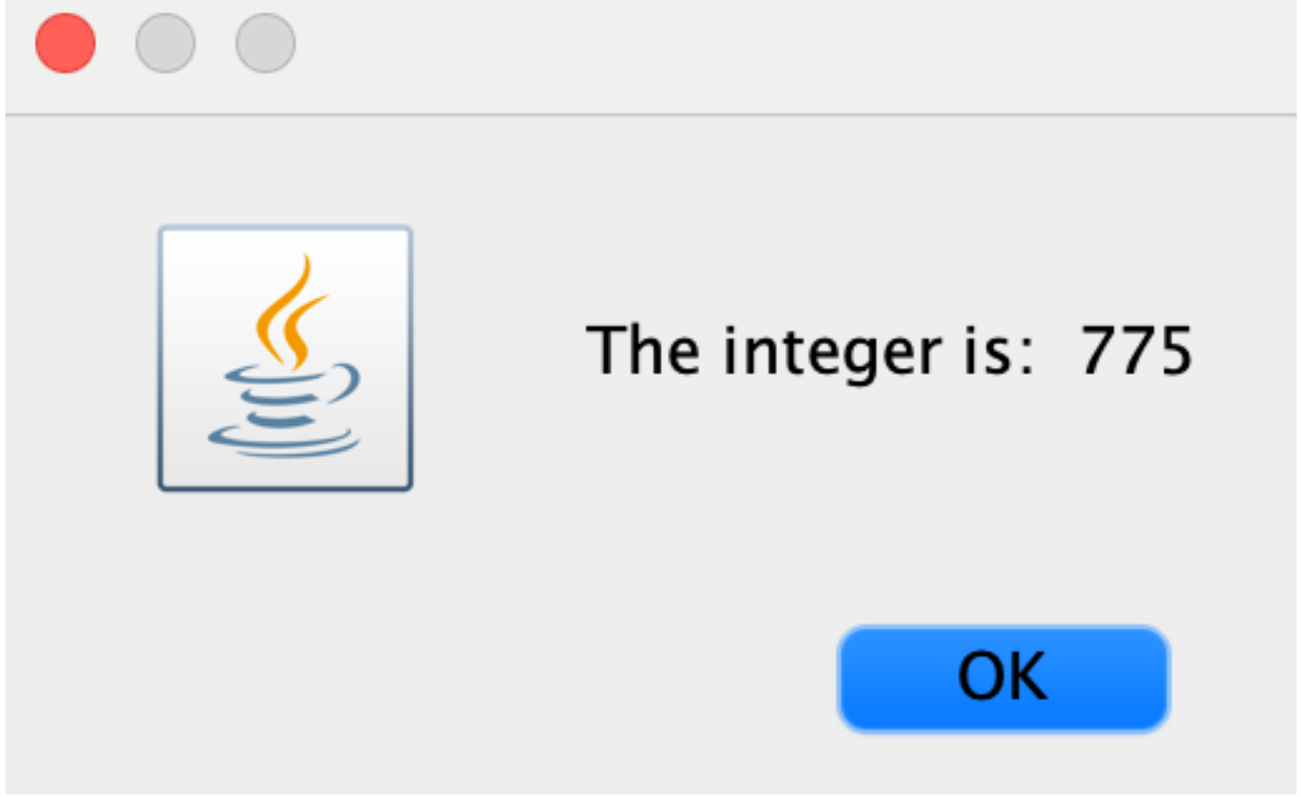
Example:

```
.data
message: .asciiz "The integer is:  "

.text
        li $v0, 56
        la $a0, message
        li $a1, 0x307   # the interger to be printed is 0x307
        syscall         # execute
```

2. MessageDialogInt

- Hiển thị một số nguyên dưới dạng hộp thoại:

- Kết quả:



3. In chuỗi

- Đưa ra đầu ra chuẩn (màn hình) một chuỗi đã định dạng.

Argument(s):

$v0 = 4
$a0 = value to be printed

Return value:

none

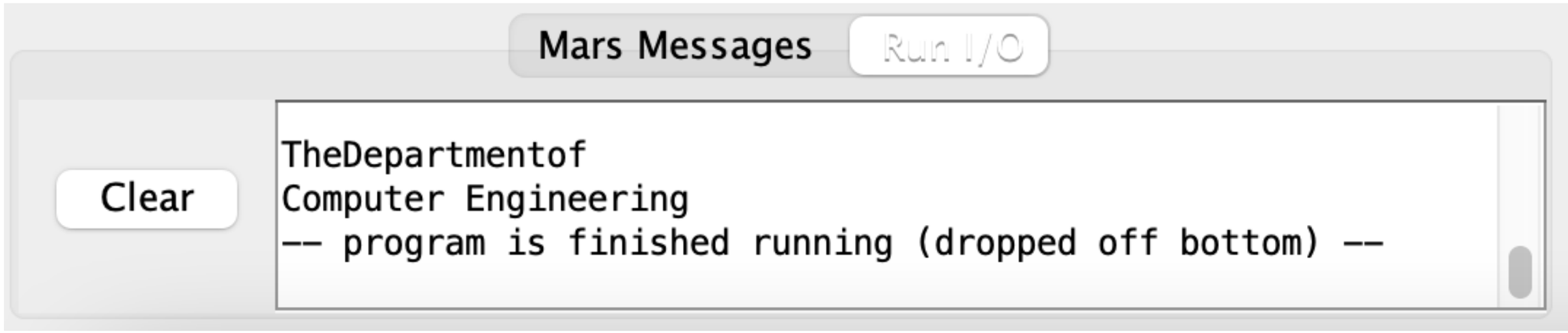
Example:

```
.data
message: .asciiz "TheDepartmentof \nComputer Engineering"

.text
        li $v0, 4
        la $a0, message
        syscall
```

3. In chuỗi

- Đưa ra đầu ra chuẩn (màn hình) một chuỗi đã định dạng.

- Kết quả:



4. MessageDialogString

- Hiển thị một chuỗi ra dưới dạng hộp thoại:

Argument(s):

$v0 = 59
$a0 = address of the null-terminated message string
$a1 = address of null-terminated string to display

Return value:

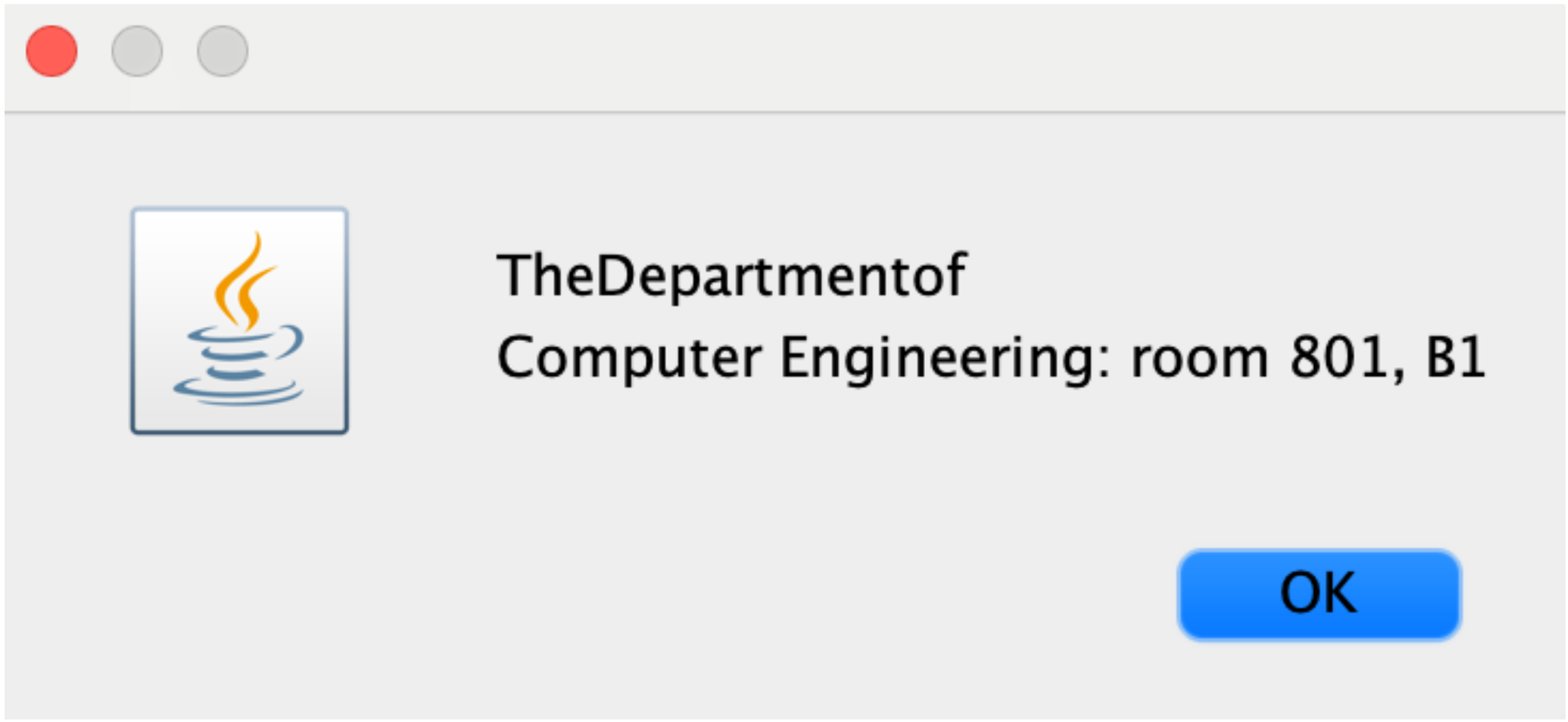
none

Example:

```
.data
message: .asciiz "TheDepartmentof \nComputer Engineering:"
address: .asciiz " room 801, B1"

.text
        li $v0, 59
        la $a0, message
        la $a1, address
        syscall
```

4. MessageDialogString

- Hiển thị một chuỗi ra dưới dạng hộp thoại.
- Kết quả:



5. Đọc số nguyên

- Đọc vào một số nguyên từ đầu vào chuẩn (bàn phím).

Argument(s):

$v0 = 5

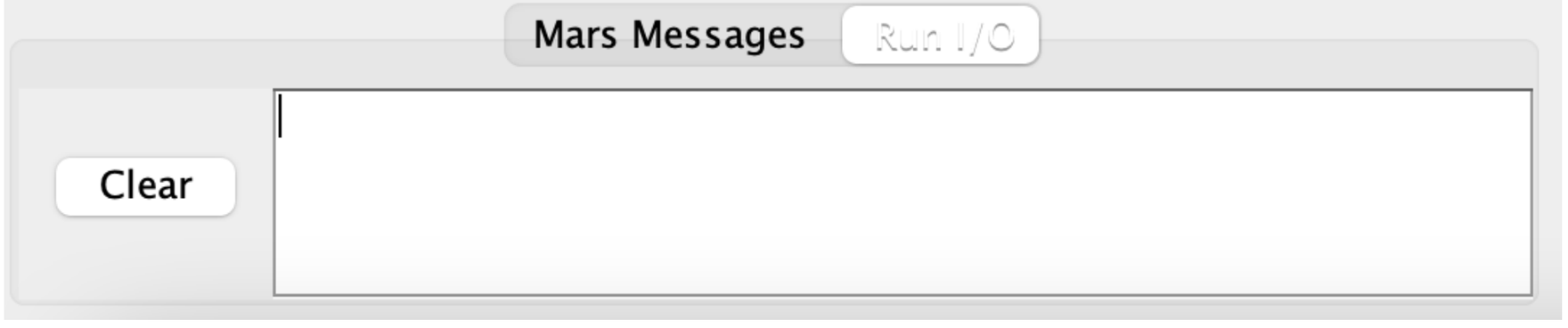
Return value:

$v0 = contains integer read

Example:

```
li $v0, 5
syscall
```

- Kết quả:



6. InputDialogInt

- Hiển thị hộp thoại thông báo để đọc một số nguyên:

Argument(s):

$v0 = 51
$a0 = address of the null-terminated message string

Return value:

$a0 = contains int read
$a1 contains status value
0: OK status
-1: input data cannot be correctly parsed
-2: Cancel was chosen
-3: OK was chosen but no data had been input into field

6. InputDialogInt

- Show a message dialog to read a integer with content parser.

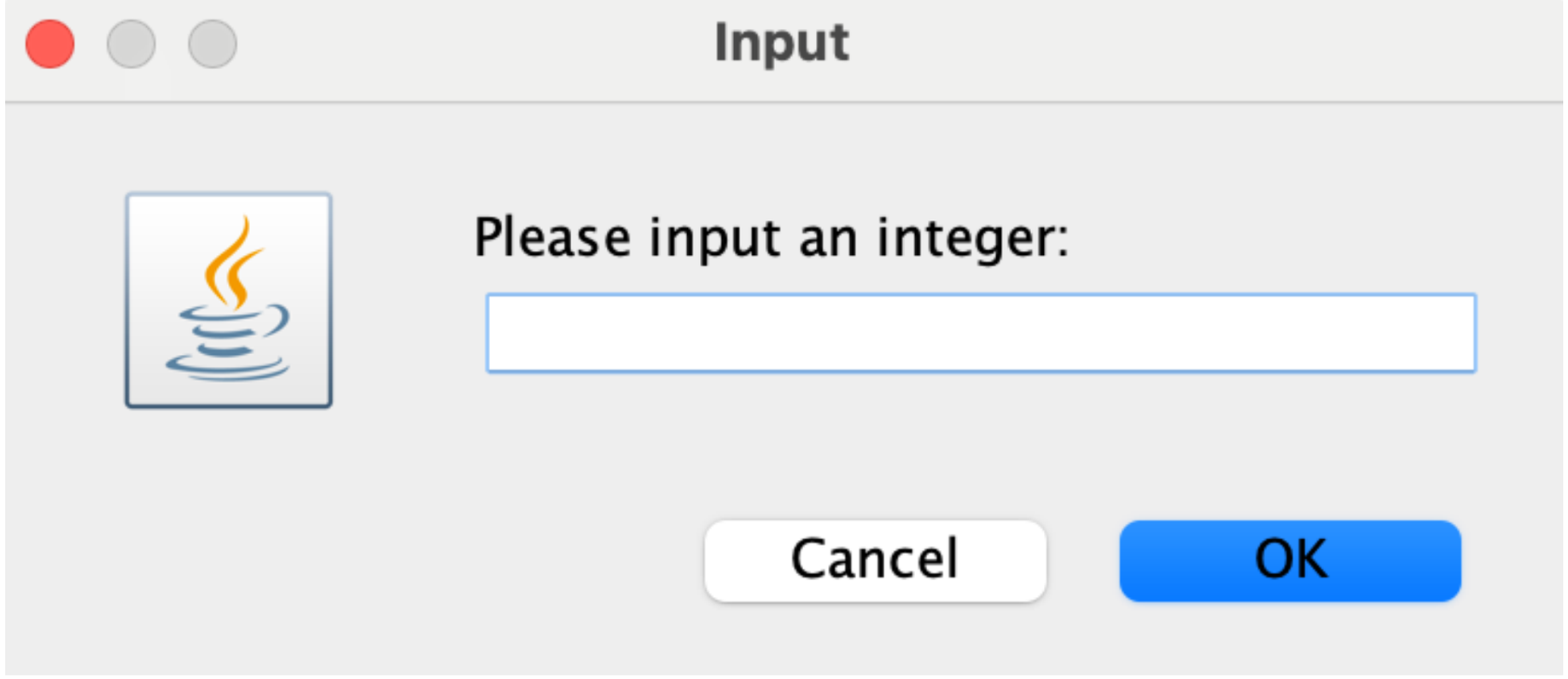
- Example:

```
.data
message: .asciiz "Please input an integer: "

.text
        li $v0, 51
        la $a0, message
        syscall
```

6. InputDialogInt

- Hiển thị hộp thoại thông báo để đọc một số nguyên:

- Kết quả:



7. Đọc chuỗi

- Đọc một chuỗi từ đầu vào chuẩn (bàn phím).

Argument(s):

$v0 = 8
$a0 = address of input buffer
$a1 = maximum number of characters to read

Return value:

none

Remarks:

For specified length n, string can be no longer than n-1.

- If less than that, adds newline to end.
- In either case, then pads with null byte

Just in special cases:

If n = 1, input is ignored and null byte placed at buffer address.

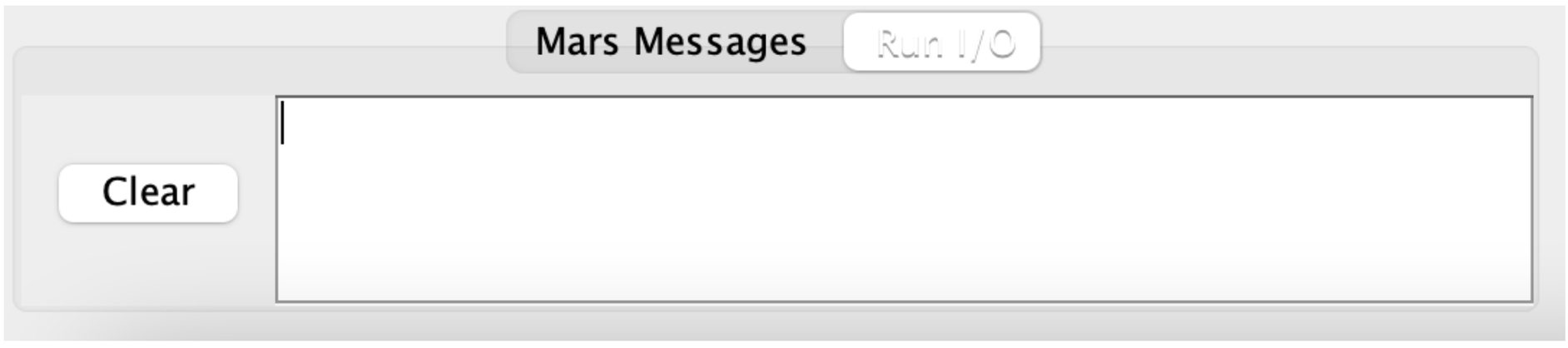
If n < 1, input is ignored and nothing is written to the buffer.

7. Đọc chuỗi

- Đọc một chuỗi từ đầu vào chuẩn (bàn phím).
- Ví dụ:

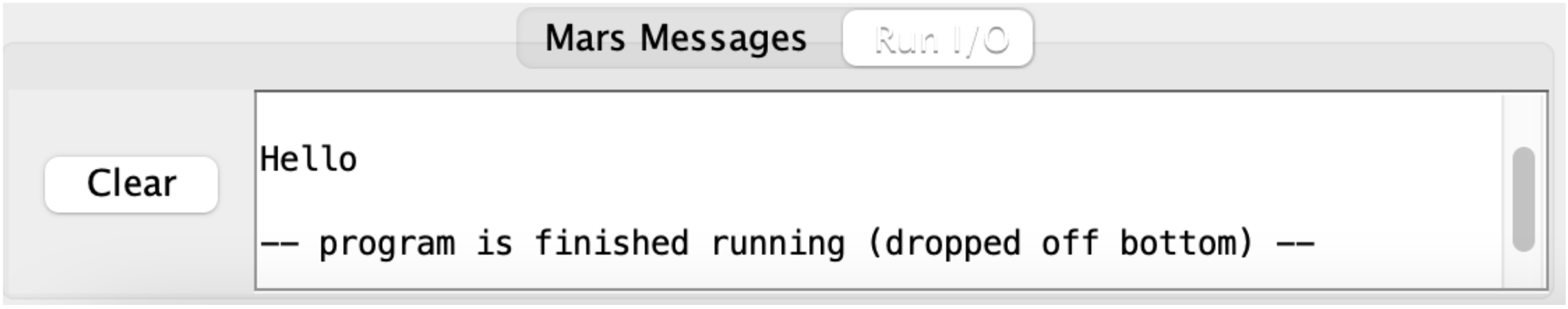
```
.data
message: .space 100     # Buffer 100 bytes for the input string

.text
        li $v0, 8
        la $a0, message
        li $a1, 100
        syscall
```

- Kết quả

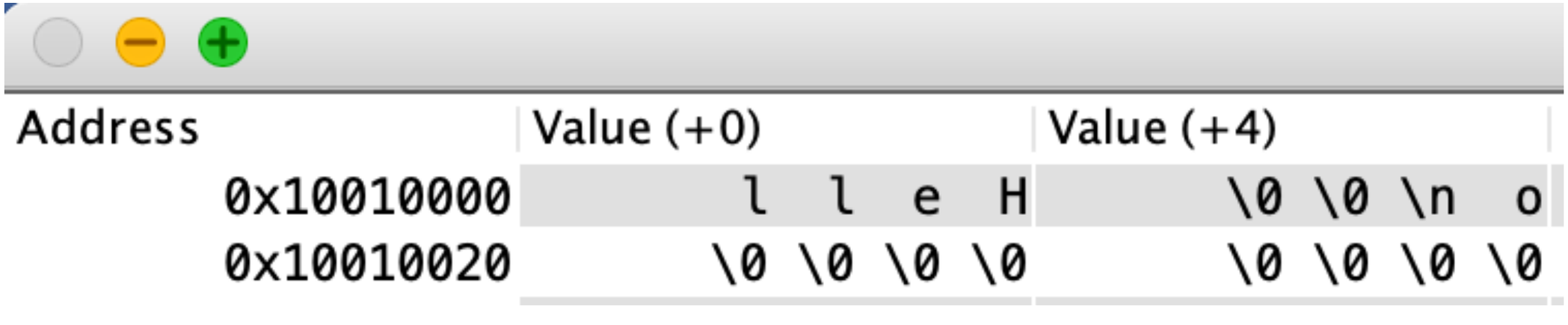


7. Đọc chuỗi

- Đọc một chuỗi từ đầu vào chuẩn (bàn phím).
- Nhập vào chuỗi:



- Kết quả



8. InputDialogString

- Show a message dialog to read a string with content parser.

Argument(s):

$v0 = 54
$a0 = address of the null-terminated message string
$a1 = address of input buffer
$a2 = maximum number of characters to read

Return value:

$a1 contains status value
0: OK status
-2: OK was chosen but no data had been input into field.
No change to buffer.
-3: OK was chosen but no data had been input into field
-4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null.

8. InputDialogString

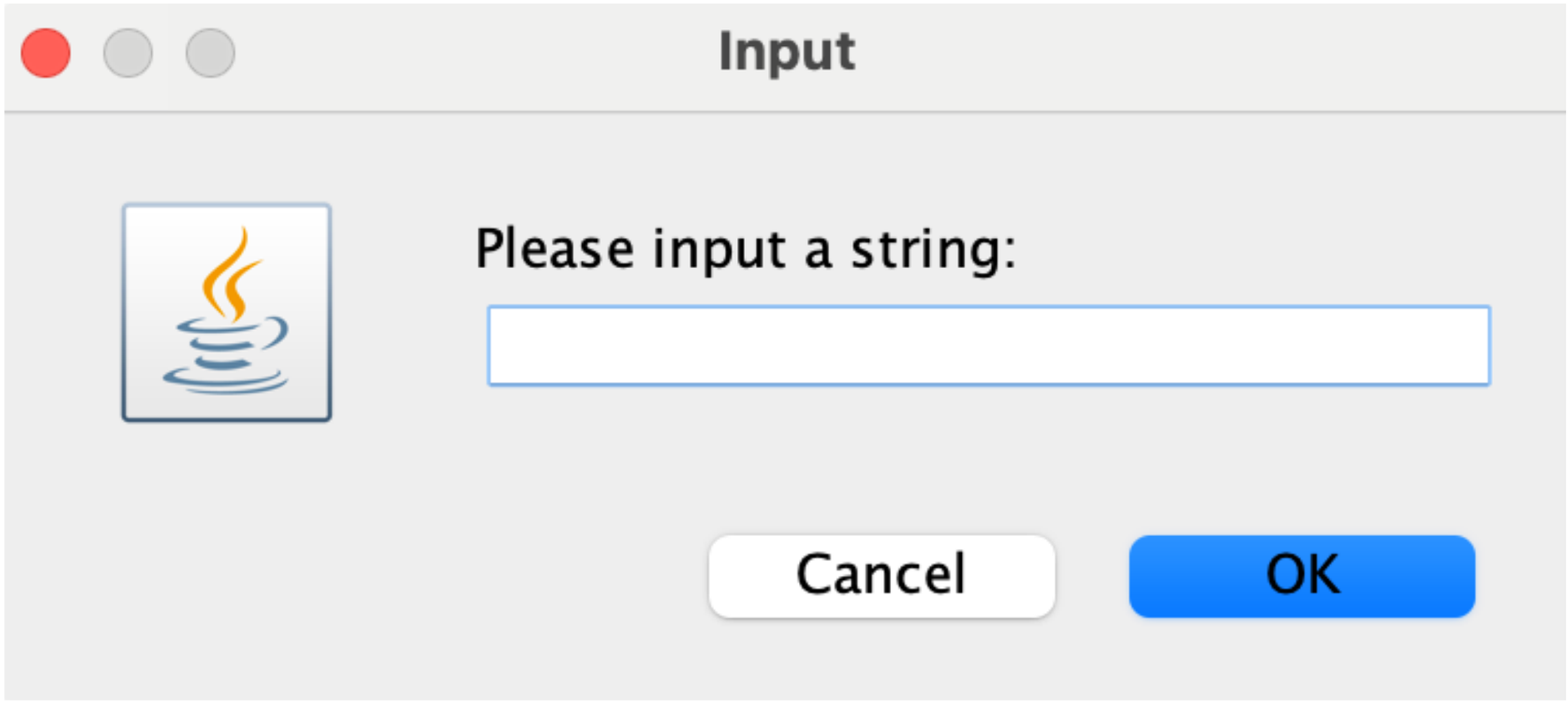
- Hiển thị hộp thoại để đọc vào một chuỗi.

- Ví dụ:

```
.data
message: .asciiz "Please input a string: "
string: .space 100

.text
        li $v0, 54
        la $a0, message
        la $a1, string
        la $a2, 100
        syscall
```

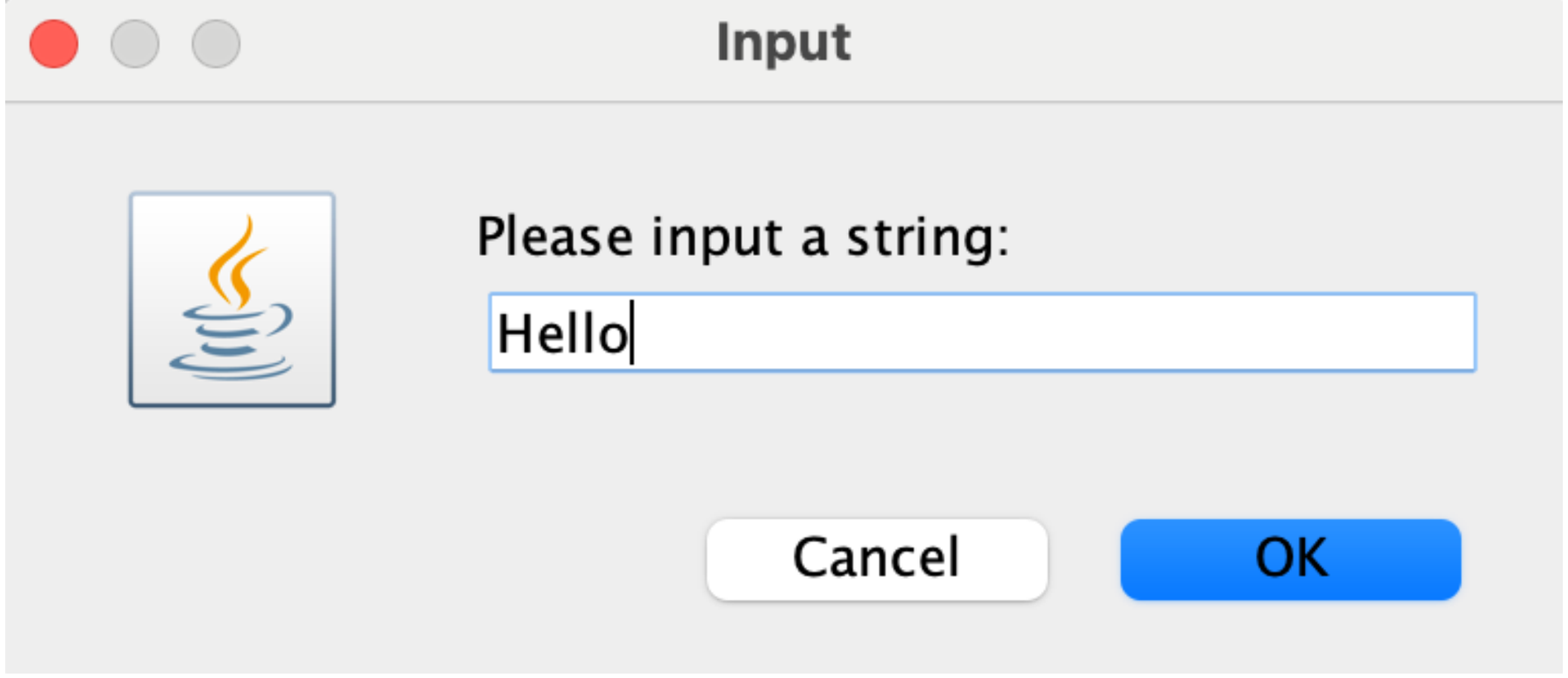
8. InputDialogString

- Hiển thị hộp thoại để đọc vào một chuỗi.
- Kết quả:



8. InputDialogString

- Hiển thị hộp thoại để đọc vào một chuỗi.
- Chuỗi nhập vào:



8. InputDialogString

- Hiển thị hộp thoại để đọc vào một chuỗi.
- Kết quả:

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	a e l P	i e s	t u p n	s a	n i r t	\0 : g	l l e H	\0 \0 \n o
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

Value (+18)	Value (+1c)
l l e H	\0 \0 \n o
\0 \0 \0 \0	\0 \0 \0 \0

9. In ký tự

- In một ký tự ra đầu ra chuẩn (màn hình)

Argument(s):

$v0 = 11
$a0 = character to print (at the lowest significant byte)

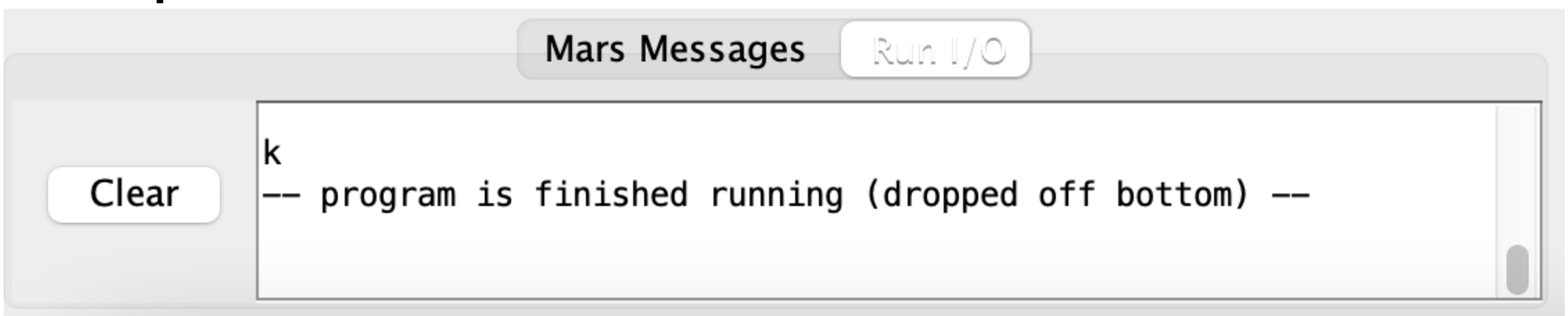
Return value:

none

Example:

```
li $v0, 11
li $a0, 'k'
syscall
```

- Kết quả



10. Đọc ký tự

- Nhận một ký tự từ đầu vào chuẩn (bàn phím)

Argument(s):

$v0 = 12

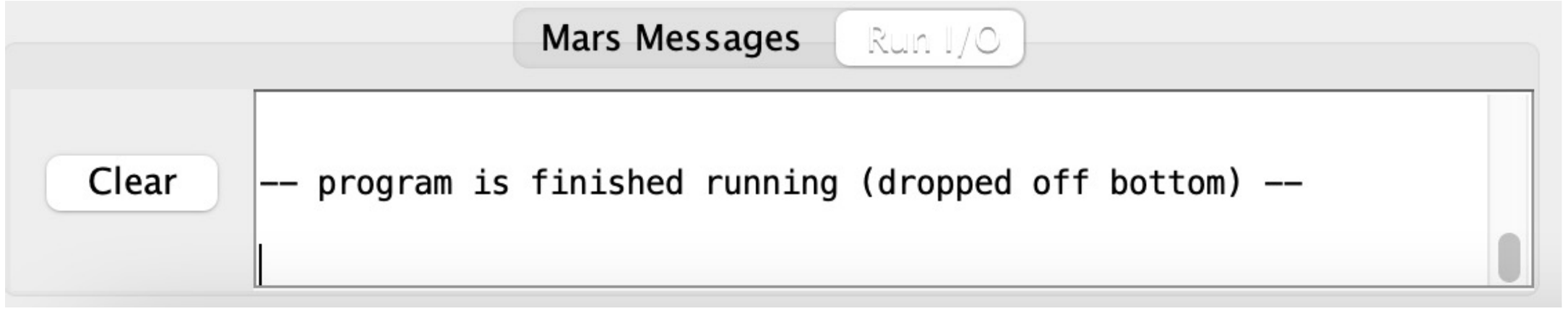
Return value:

$v0 contains character read

Example:

```
li $v0, 12
syscall
```

- Kết quả



11. Hộp thoại xác nhận

- Hiển thị một hộp thoại với 3 nút: Yes | No | Cancel

Argument(s):

$v0 = 50
$a0 = address of the null-terminated message string

Return value:

$a0 = contains value of user-chosen option
0: Yes
1: No
2: Cancel

11. Hộp thoại xác nhận

- Hiển thị một hộp thoại với 3 nút: Yes | No | Cancel

- Ví dụ:

```
.data
message: .asciiz "Do you agree?"

.text
        li $v0, 50
        la $a0, message
        syscall
```

11. Hộp thoại xác nhận

- Hiển thị một hộp thoại với 3 nút: Yes | No | Cancel

- Kết quả

12. Hộp thoại thông tin

- Hiển thị một hộp thoại thông tin chỉ với biểu tượng và nút OK.

Argument(s):

$v0	= 55
$a0	= address of the null-terminated message string
$a1	= the type of message to be displayed: 0: error message, indicated by Error icon 1: information message, indicated by Information icon 2: warning message, indicated by Warning icon 3: question message, indicated by Question icon other: plain message (no icon displayed)

Return value:

none

12. Hộp thoại thông tin

- Hiển thị một hộp thoại thông tin chỉ với biểu tượng và nút OK.

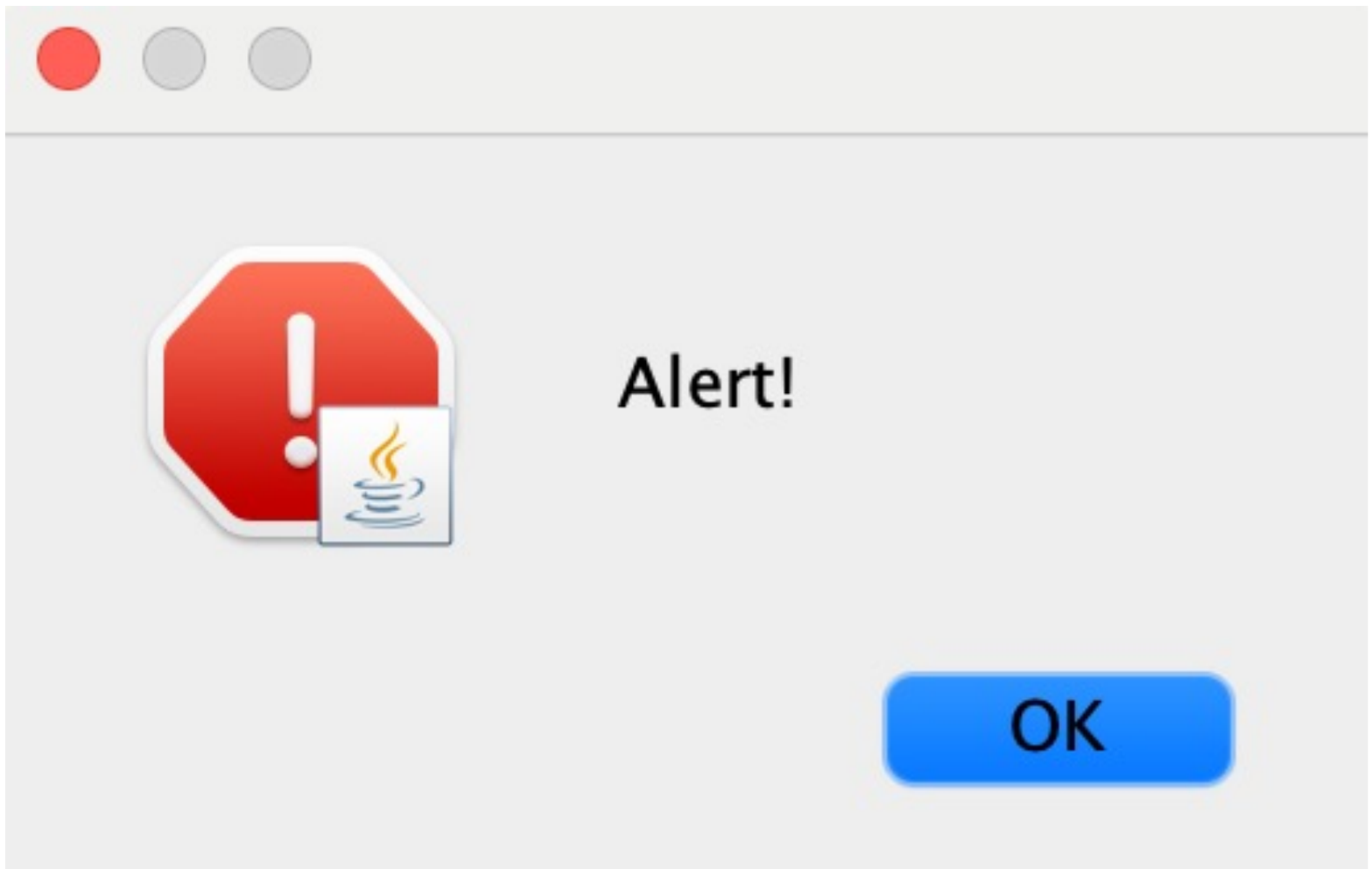
- Ví dụ:

```
.data
message: .asciiz "Alert!"

.text
        li $v0, 55
        la $a0, message
        syscall
```

12. Hộp thoại thông tin

- Hiển thị một hộp thoại thông tin chỉ với biểu tượng và nút OK.

- Kết quả



Sử dung dịch vụ SYSCALL 31 và 33 đầu ra MIDI

- Các dịch vụ hệ thống trên là duy nhất trên MARS và cung cấp một phương pháp tạo ra âm thanh.

- Đầu ra MIDI được mô phỏng card âm thanh của hệ thống và mô phỏng được thực hiện bởi javax.sound.midipackage

  - Dịch vụ 31 sẽ tạo ra một âm báo, sau đó trả về ngay lập tức.

  - Dịch vụ 33 sẽ tạo ra một âm báo, sau đó ngủ trong toàn bộ thời gian của giai điệu và trả về. Do đó, thường là kết hợp dịch vụ 31 và 33 với nhau.

- Dịch vụ này cần 4 tham số như sau:

Sử dụng dịch vụ SYSCALL 31 và 33
Đầu ra MIDI

pitch ($a0)

- Accepts a positive byte value (0-127) that denotes a pitch as it would be represented in MIDI
- Each number is one semitone / half-step in the chromatic scale.
- 0 represents a very low C and 127 represents a very high G (a standard 88 key piano begins at 9-A and ends at 108-C).
- If the parameter value is outside this range, it applies a default value 60 which is the same as middle C on a piano.
- From middle C, all other pitches in the octave are as follows:
  - 61 = C# or Db
  - 62 = D
  - 63 = D# or Eb
  - 64 = E or Fb
  - 65 = E# or F
  - 66 = F# or Gb
  - 67 = G
  - 68 = G# or Ab
  - 69 = A
  - 70 = A# or Bb
  - 71 = B or Cb
  - 72 = B# or C
- To produce these pitches in other octaves, add or subtract multiples of 12.

Sử dụng dịch vụ SYSCALL 31 và 33 Đầu ra MIDI

duration in milliseconds ($a1)

- Accepts a positive integer value that is the length of the tone in milliseconds.
- If the parameter value is negative, it applies a default value of one second (1000 milliseconds).

Sử dụng dịch vụ SYSCALL 31 và 33
Đầu ra MIDI

instrument ($a2)

- Accepts a positive byte value (0-127) that denotes the General MIDI "patch" used to play the tone.
- If the parameter is outside this range, it applies a default value 0 which is an *Acoustic Grand Piano*.
- General MIDI standardizes the number associated with each possible instrument (often referred to as *program change* numbers), however it does not determine how the tone will sound. This is determined by the synthesizer that is producing the sound. Thus a *Tuba* (patch 58) on one computer may sound different than that same patch on another computer.
- The 128 available patches are divided into instrument families of 8:

0-7	Piano	64-71	Reed
8-15	Chromatic Percussion	72-79	Pipe
16-23	Organ	80-87	Synth Lead
24-31	Guitar	88-95	Synth Pad
32-39	Bass	96-103	Synth Effects
40-47	Strings	104-111	Ethnic
48-55	Ensemble	112-119	Percussion
56-63	Brass	120-127	Sound Effects

- Note that outside of Java, General MIDI usually refers to patches 1-128. When referring to a list of General MIDI patches, 1 must be subtracted to play the correct patch. For a full list of General MIDI instruments, see www.midi.org/about-midi/gm/gm1sound.shtml. The General MIDI channel 10 percussion key map is not relevant to the toneGenerator method because it always defaults to MIDI channel 1.

Sử dụng dịch vụ SYSCALL 31 và 33 Đầu ra MIDI

volume ($a3)

- Accepts a positive byte value (0-127) where 127 is the loudest and 0 is silent. This value denotes MIDI velocity which refers to the initial attack of the tone.
- If the parameter value is outside this range, it applies a default value 100.
- MIDI velocity measures how hard a *note on* (or *note off*) message is played, perhaps on a MIDI controller like a keyboard. Most MIDI synthesizers will translate this into volume on a logarithmic scale in which the difference in amplitude decreases as the velocity value increases.
- Note that velocity value on more sophisticated synthesizers can also affect the timbre of the tone (as most instruments sound different when they are played louder or softer).

13. MIDI out

- Tạo ra một âm báo

Argument(s):

$v0 = 31
$a0 = pitch (0-127)
$a1 = duration in milliseconds
$a2 = instrument (0-127)
$a3 = volume (0-127)

Return value:

Generate tone and return immediately

13. MIDI out

- Tạo ra một âm báo
- Ví dụ:

```
li $v0, 31
li $a0, 42          #pitch
li $a1, 2000        #time
li $a2, 0           #musical instrusment
li $a3, 212         #volume
syscall
```

14. Đồng bộ MIDI out

- Tạo một âm báo

Argument(s):

$v0 = 33
$a0 = pitch (0-127)
$a1 = duration in milliseconds
$a2 = instrument (0-127)
$a3 = volume (0-127)

Return value:

Generate tone and return upon tone completion

14. MIDI out synchronous

- Tạo một âm báo
- Ví dụ:

```
li $v0, 33
li $a0, 42      #pitch
li $a1, 2000    #time
li $a2, 0       #musical instrusment
li $a3, 212     #volume
syscall
```

15. Thoát

- Kết thúc chương trình

  - Không có lệnh EXIT trong tập lệnh của bất kỳ bộ vi xử lý nào.

  - Exit là một dịch vụ thuộc Hệ điều hành.

Argument(s):

$v0 = 10

Return value:

none

15. Thoát

- Kết thúc chương trình

  - Không có lệnh EXIT trong tập lệnh của bất kỳ bộ vi xử lý nào.

  - Exit là một dịch vụ thuộc Hệ điều hành.

- Ví dụ:

```
li $v0, 10 #exit
syscall
```

16. Exit with code

- Kết thúc chương trình

  - Không có lệnh EXIT trong tập lệnh của bất kỳ bộ vi xử lý nào.

  - Exit là một dịch vụ thuộc Hệ điều hành.

Argument(s):

$v0 = 17
$a0 = termination result

Return value:

none

16. Exit with code

- Kết thúc chương trình

  - Không có lệnh EXIT trong tập lệnh của bất kỳ bộ vi xử lý nào.

  - Exit là một dịch vụ thuộc Hệ điều hành.

- Ví dụ:

```
li $v0, 17 # exit
li $a0, 3  # with error code = 3
syscall
```

Bài tập 7.2

- Chương trình hợp ngữ đơn giản sau sẽ hiển thị một chuỗi chào mừng.

  - Sử dụng hàm printf cho mục đích này.

  - Đọc kỹ ví dụ này, chú ý cách truyền tham số cho hàm **printf**.

```
.data
test: .asciiz "Hello World"

.text
        li  $v0, 4
        la  $a0, test
        syscall
```

Bài tập 7.2

- Tạo một dự án mới để thực hiện chương trình trên.

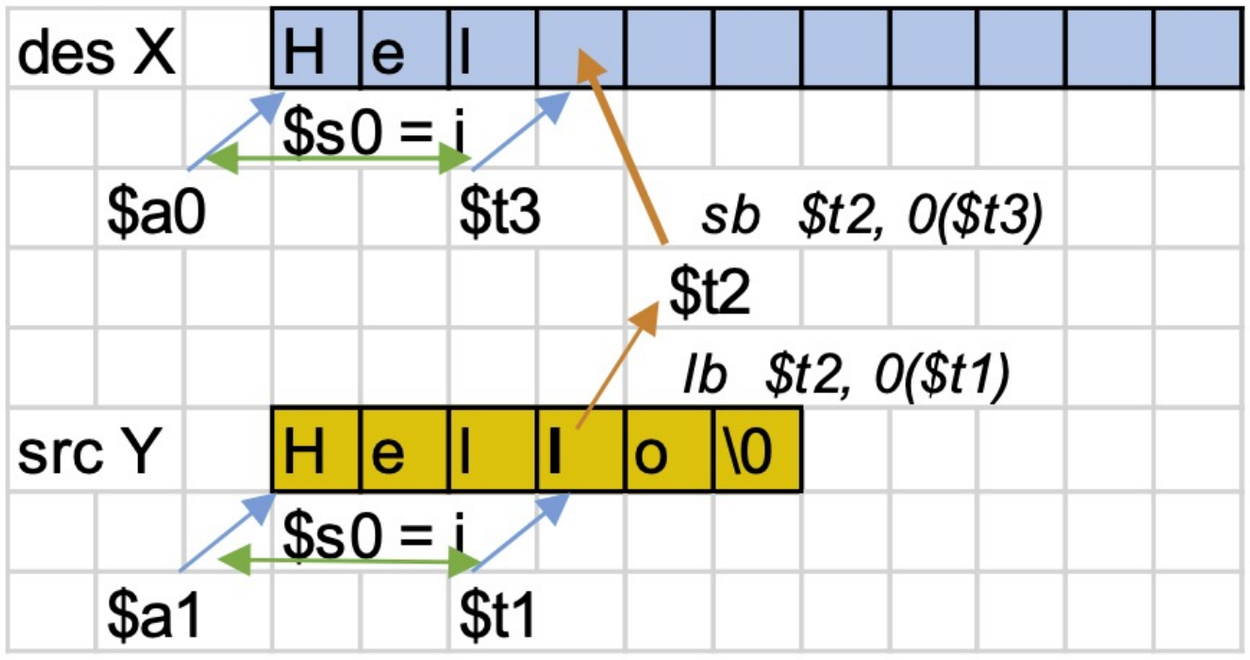
- Biên dịch và thực hiện mô phỏng.

- Chạy và quan sát kết quả.

- Vào phần bộ nhớ dữ liệu, kiểm tra cách chuỗi được lưu trữ và đóng gói trong bộ nhớ như thế nào.

Bài tập 7.3

- Hàm **strcpy** copy **string y** sang **string x** sử dụng sử dụng quy ước kết thúc byte rỗng của C.
- Đọc kỹ ví dụ này, cố gắng hiểu tất cả phần mã này.



Bài tập 7.3

```
.data
x: .space 1000              # destination string x, empty
y: .asciiz "Hello"          # source string y

.text
strcpy:
        add $s0,$zero,$zero                 #s0 = i=0
L1:
        add $t1,$s0,$a1                     #t1 = s0 + a1 = i + y[0]
                                            # = address of y[i]
        lb $t2,0($t1)                       #t2 = value at t1 = y[i]
        add $t3,$s0,$a0                     #t3 = s0 + a0 = i + x[0]
                                            # = address of x[i]
        sb $t2,0($t3)                       #x[i]= t2 = y[i]
        beq $t2,$zero,end_of_strcpy         #if y[i]==0, exit
        nop
        addi $s0,$s0,1                      #s0=s0 + 1 <-> i=i+1
        j L1                                #next character
        nop
end_of_strcpy:
```

Bài tập 7.3

- Tạo một dự án mới để in tổng của hai thanh ghi **$s0** và **$s1** theo định dạng sau:

  “Tổng của (s0) và (s1) là (kết quả)”

Bài tập 7.4

- Chương trình sau đây đếm độ dài của một chuỗi kết thúc bằng null.
- Đọc kỹ ví dụ này, phân tích từng dòng mã.

Bài tập 7.4

```
.text
main:
get_string:                                 # TODO
get_length:
        la $a0, string                      # a0 = Address(string[0])
        xor $v0, $zero, $zero               # v0 = length = 0
        xor $t0, $zero, $zero               # t0 = i = 0

check_char:
        add $t1, $a0, $t0                   # t1 = a0 + t0
                                            #= Address(string[0]+i)
        lb   $t2, 0($t1)                    # t2 = string[i]
        beq  $t2,$zero,end_of_str           # Is null char?
        addi $v0, $v0, 1                    # v0=v0+1->length=length+1
        addi $t0, $t0, 1                    # t0=t0+1->i = i + 1
        j    check_char
end_of_str:
end_of_get_length:
print_length:                               # TODO
```

Bài tập 7.4

- Tạo một dự án mới để thực hiện chương trình trên.

  - Thêm các lệnh khác để gán một chuỗi cho biến y và thực hiện hàm **strcpy**.

- Biên dịch và thực hiện mô phỏng.

- Chạy và quan sát kết quả.

Bài tập 7.5

- Thực hiện bài 7.4 với hàm để:
  - Nhận một chuỗi từ hộp thoại
  - Hiển thị chiều dài lên hộp thoại thông tin cho hộp thoại tin nhắn

Bài tập 7.6

- Viết chương trình cho phép người dùng nhập vào một chuỗi.

  - Quá trình nhập liệu sẽ bị chấm dứt khi người dùng nhấn Enter hoặc độ dài của chuỗi vượt quá 20 ký tự.

  - In chuỗi ngược lại.

Bài tập 7.7

- Viết một chương trình cho phép người dùng đoán một số ngẫu nhiên do máy tính tạo ra từ 1 đến cực đại (người dùng nhập giá trị lớn nhất để đoán).

  - Trong chương trình này, người dùng sẽ nhập giá trị của giá trị lớn nhất và syscall 42 sẽ được sử dụng để tạo ra một số ngẫu nhiên từ 1 đến lớn nhất.

  - Sau đó, người dùng sẽ nhập các dự đoán và chương trình sẽ in ra nếu kết quả đoán quá cao hoặc quá thấp cho đến khi người dùng đoán đúng số.

  - Chương trình sẽ in ra số lần đoán mà người dùng đã thực hiện.

Kết thúc tuần 7