

CHƯƠNG 4: NGÔN NGỮ HỢP DỊCH(HỢP NGỮ-ASSEMBLY)

4.1 Các khái niệm:

Hợp ngữ là ngôn ngữ lập trình cho máy tính nằm giữa ngôn ngữ máy và ngôn ngữ cấp cao như C, Pascal.

Hợp ngữ có dạng văn bản thay cho ngôn ngữ máy(mã nhị phân) gồm tập hợp các từ gọi nhớ dễ viết và đọc và dễ nhớ hơn các mã máy dưới dạng nhị phân.Ví dụ từ gọi nhớ hay lệnh **ADD R16,R17** thay cho mã máy **0000111100000001B** thực hiện tác vụ cộng nội dung 2 thanh ghi R16 và R7.

Chương trình viết bằng hợp ngữ theo các quy tắc nhất định được biên dịch thành mã máy, MCU chỉ hiểu và thực hiện chương trình dưới dạng mã máy. Trình biên dịch hợp ngữ sang mã máy tương ứng gọi là trình hợp dịch(assembler).

Ngoài ra còn có trình liên kết/dịnh vị lại(linker/relocator) để kết nối các phần của chương trình nằm trên các file khác nhau và sắp xếp lại địa chỉ bộ nhớ tạo thành 1 chương trình thực thi được.

Các định nghĩa:

- **Chương trình hợp ngữ(assembly language program)** là chương trình sử dụng nhãn, từ gọi nhớ...trong đó mỗi phát biểu tương đương 1 lệnh ngôn ngữ máy. Chương trình hợp ngữ thường gọi là mã nguồn hay mã ký hiệu, máy tính không thể thực thi được.

- **Chương trình ngôn ngữ máy(machine language program)** là chương trình chứa các mã nhị phân biểu diễn các lệnh máy tính thực thi. Chương trình ngôn ngữ máy thường gọi là mã đối tượng(object code) máy tính thực thi được.

- **Trình hợp dịch(assembler)** chuyển đổi chương trình hợp ngữ sang chương trình ngôn ngữ máy (object code) dưới dạng tuyệt đối hoặc định vị lại(relocator). Trường hợp sau cần phải có sự liên kết để tạo thành địa chỉ tuyệt đối mới thực thi được.

- **Trình liên kết(linker)** kết nối các chương trình ngôn ngữ máy định vị lại(mô đun) tạo thành một chương trình ngôn ngữ máy tuyệt đối máy tính có thể thực thi được. Trình liên kết còn được gọi là liên kết /dịnh vị lại(linker/relocator) thể hiện chức năng sắp xếp kết hợp các mô đun và đặt địa chỉ tuyệt đối để thực thi.

- **Phân đoạn(segment)** là một đơn vị của bộ nhớ code hoặc data. Phân đoạn có thể có dạng định vị lại hoặc tuyệt đối. Một phân đoạn định vị lại có tên, loại và các thuộc tính cho phép trình liên kết kết hợp nó với các phân đoạn khác nếu có yêu cầu, và đặt đúng vị trí phân đoạn. Một phân đoạn tuyệt đối không có tên và không thể kết hợp với các phân đoạn khác.

- **Mô đun(module)** chứa một hoặc nhiều phân đoạn hoặc các phân đoạn không hoàn chỉnh. Mô đun có tên do người sử dụng đặt. Mỗi mô đun phải định nghĩa xác định các ký hiệu sử dụng cục bộ(local). Một file object có thể chứa một hoặc nhiều mô đun. Đôi khi mô đun có thể gọi là file.

- **Chương trình(program)** chứa một mô đun tuyệt đối, kết hợp tất cả các phân đoạn tuyệt đối và định vị lại từ tất cả các mô đun ngõ vào. Một chương trình chỉ chứa mã máy thực thi lệnh(gồm các địa chỉ và data hằng) và máy tính có thể hiểu và thực thi.

4.2 Định dạng chương trình hợp ngữ:

- Một chương trình hợp ngữ gồm các phần sau:

- Các lệnh.
- Các chỉ dẫn biên dịch (assembler directives).
- Các điều khiển biên dịch (assembler controls).
- Các ghi chú giải thích (comments).

- Định dạng một dòng lệnh:

[Nhãn:] **Từ gọi nhớ [Toán hạng 1][,Toán hạng 2] [...] [;Chú giải]**

- Các thành phần trong một dòng lệnh gọi là các trường (field).
- Các trường cách nhau ít nhất một dấu cách(space bar) hay 1 tab(trừ trường chú giải).
- Trường từ gọi nhớ là bắt buộc phải có trong một dòng lệnh.
- Trường nhãn có thể đặt trên, không cần cùng dòng với trường từ gọi nhớ.
- Các trường toán hạng phải cùng dòng với trường từ gọi nhớ.

Ví dụ 4.1:

START: INC R16 ;tăng R16 lên 1
RJMP START ;nhảy đến nhãn START

Hoặc

START:

```
INC R16 ;tăng R16 lên 1  
RJMP START ;nhảy đến nhãn START
```

4.2.1 Trường nhãn:

- Luôn đặt ở đầu dòng lệnh, biểu diễn địa chỉ của lệnh theo sau.
- Nhãn chính là toán hạng trong các lệnh rẽ nhánh đến lệnh có nhãn đứng trước nó.
- Phân biệt nhãn với ký hiệu: **sau nhãn có dấu hai chấm :**
- Một nhãn chỉ biểu diễn duy nhất một địa chỉ, không được dùng cho 2 dòng lệnh khác nhau.
- Tên của nhãn có thể là các ký tự chữ cái(gồm chữ thường và chữ hoa), các số từ 0-9, hoặc ký tự đặc biệt là dấu gạch dưới (_). Ký tự đầu tiên của nhãn bắt buộc phải là ký tự chữ cái, hoặc dấu gạch dưới (_), nhãn không được phép trùng với từ gợi nhớ, chỉ dẫn, từ điều khiển hoặc ký hiệu đã định nghĩa.

Ví dụ 4.2:

```
.EQU MYCOUNT 15 ;định nghĩa hằng số MYCOUNT=15  
LOOP: LDI R17,$55 ;nhãn LOOP đặt ngay dòng lệnh LDI
```

4.2.2 Trường từ gọi nhớ:

- Các ký hiệu lệnh hoặc chỉ dẫn nằm trong trường từ gọi nhớ.
- Trường từ gọi nhớ đặt sau trường nhãn.
- Các ký hiệu lệnh như: **ADD, SUB, AND, OR, MUL, MOV, LD...**
- Các ký hiệu chỉ dẫn như: **ORG, EQU, DB, BIT, ...**

4.2.3 Trường toán hạng:

- Theo sau trường từ gọi nhớ chứa địa chỉ hoặc dữ liệu thực hiện lệnh.
- Có thể là nhãn hoặc ký hiệu định nghĩa hằng số.
- Có thể không xuất hiện trường toán hạng tùy vào lệnh ví dụ: NOP, RET, RETI..
- Có thể có 1 hoặc nhiều trường toán hạng trong cùng 1 dòng lệnh, cách nhau bởi dấu phẩy.

Ví dụ 4.3:

```
.DEF COUNT = R20 ; COUNT là biến đếm có giá trị là nội dung của R20.  
...  
DELAY: LDI COUNT, 0xFF ;  
AGAIN: NOP ; lệnh không toán hạng  
        NOP ;  
        DEC COUNT ; COUNT là ký hiệu  
        BRNE AGAIN ; AGAIN là 1 nhãn  
        RET ; lệnh không toán hạng
```

4.2.4 Trường chú giải:

- Phải có dấu chấm phẩy (;) đặt trước, khi biên dịch trình biên dịch sẽ loại bỏ các ký tự sau dấu chấm phẩy.
- Khi soạn thảo chương trình nên ghi chú giải thích ý nghĩa thực hiện các dòng lệnh, đoạn chương trình hoặc chương trình con để dễ hiểu khi đọc lại.

4.3 Các toán tử và biểu thức với các toán hạng là hằng số:

- **Định dạng dữ liệu:**

- Vì điều khiển AVR chỉ sử dụng một kiểu dữ liệu là 8bit(1byte) và kích thước của mỗi thanh ghi cũng là 8bit.
 - Có 4 cách để biểu diễn dữ liệu trong AVR: số nhị phân(binary), thập lục phân(hex), thập phân(decimal) hoặc mã ASCII.

Số Hex - Có 2 cách để biểu diễn: dùng ký hiệu **0x** hoặc dấu **\$**.

VD: LDI R16,**0x99** Hoặc LDI R16,\$99

Số nhị phân: sử dụng ký hiệu **0b**.

VD: LDI R16,**0b10011001**; R16 = 10011001B = 99H

Số thập phân: không sử dụng ký hiệu kèm theo.

VD: SUBI R17,**2** ; R17=R17- 2

Mã ASCII: ký tự được đặt trong 2 dấu nháy đơn ''.

VD: LDI R22,'**2**' ; R22=32

LDI R23,'A' ; R23=41

- **Các toán tử số học:**

Ký hiệu	Phép toán
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Phần dư(modulo)

Ví dụ 4.4:

LDI R16,10+0x2A ;R16=0x34=52
LDI R17,0xFF-0xF0 ;R17=0x0F=15
LDI R18,10*5 ;R18=0x32=50
LDI R19,255/5 ;R19=0x33=51
LDI R20,8%6 ;R20=0x2=2

- **Các toán tử logic:** được thực hiện trên từng bit.

Ký hiệu	Phép toán
&	AND
	OR
^	XOR
~	NOT

Ví dụ 4.5:

LDI R20,0x0F&0x75 ;R20=0x05
LDI R22,0xF0|0x75 ;R22=0xF5
LDI R24,~0x0F ;R24=0xF0
LDI R25,0x0F ^0x75 ;R25=0x7A

- **Các toán tử đặc biệt:**

Ký hiệu	Phép toán
<<	Dịch trái
>>	Dịch phải
HIGH	Byte cao
LOW	Byte thấp
()	Tính trước

Ví dụ 4.6:

LDI R20,0b01010101<<1 ;R20=0b10101010
LDI R22,0b01010101>>2 ;R22=0b00010101
LDI R24,HIGH(0xFFEE) ;R24=0xFF
LDI R25,LOW(0xFFEE) ;R25=0xEE

- **Các toán tử quan hệ:** khi sử dụng toán tử quan hệ giữa 2 toán hạng, kết quả trả về là đúng(01H) hoặc sai(00H)

Ký hiệu	Phép toán
==	Bằng
!=	Khác
<	Nhỏ hơn

\leq	Nhỏ hơn hoặc bằng
$>$	Lớn hơn
\geq	Lớn hơn hoặc bằng

Ví dụ 4.7:

```
LDI R20,5==5
LDI R20,5!=4
LDI R20,'X'<'Z'
LDI R20,'X'<= 'X'
LDI R20,'$>0
LDI R20,100 >= 50
```

Các toán tử quan hệ trên đều đúng, nên kết quả trả về trong thanh ghi R20 đều bằng **0x01**.

- **Thứ tự ưu tiên của các toán tử:** các toán tử trong biểu thức được ưu tiên từ cao xuống thấp như sau:

()	HIGH LOW
* /	
+ -	
<<, >> (DỊCH TRÁI, PHẢI)	
~ (NOT)	
& (AND)	
, ^ (OR, XOR)	

Khi các toán tử cùng mức ưu tiên thì thứ tự được thực hiện từ trái sang phải.

- Ví dụ 4.8:** Tính các biểu thức $((A-B)*2)+9$ và $(C1 \& C2)|C3$.

Với A=50, B=40, C1=0x50, C2=0x10, C3=0x04

```
LDI R20, ((50-40)*2)+9 ;R20=29
LDI R21,(0x50&0x10)|0x04 ;R21=0x14
```

4.4 Các chỉ dẫn assembler:

Các chỉ dẫn assembler là các lệnh cho trình biên dịch assembler, không phải là các lệnh cho trình hợp ngữ(assembly).

- **Định nghĩa ký hiệu:** các chỉ dẫn định nghĩa ký hiệu dùng để thay thế cho các thanh ghi, dữ liệu hoặc địa chỉ. Ký hiệu không được trùng với các định nghĩa trước đó và không được định nghĩa lại sau đó với bất kỳ hình thức nào ngoại trừ chỉ dẫn **SET**. Không cho phép nhãn đứng trước ký hiệu.
- **Chỉ dẫn INCLUDE:** chỉ dẫn này yêu cầu trình biên dịch thêm nội dung của một file cần thiết khi sử dụng với bất kỳ một AVR nào. Chẳng hạn như khi sử dụng **ATmega324P** thì file cần phải thêm vào là **M324PDEF.inc**, nội dung của file này định nghĩa tất cả các địa chỉ thanh ghi và bit được sử dụng của chip **ATmega324P**.

- Định dạng: **.INCLUDE "TÊN FILE"**
- Hoặc: **.INCLUDE <TÊN FILE>**

➤ *Lưu ý:* với một số phiên bản mới của trình biên dịch **ATMEL STUDIO7** thì không cần phải thêm chỉ dẫn này vì đã được tự động thêm vào khi biên dịch.

- **Chỉ dẫn BYTE:** dùng để định nghĩa một số byte bộ nhớ trong SRAM hoặc EEPROM. Chỉ dẫn này phải theo sau một nhãn và có tham số kèm theo.

- Định dạng: [nhãn]: **.BYTE** biểu thức

• **Chỉ dẫn CSEG:** dùng để bắt đầu cho một segment CODE. Một đoạn chương trình hay dữ liệu không khai báo thuộc segment nào thì mặc định là segment code. Chỉ dẫn BYTE sẽ không được dùng trong segment này. Chỉ dẫn này không có bất cứ tham số nào kèm theo.

- Định dạng: **.CSEG**

• **Chỉ dẫn DSEG:** dùng để bắt đầu cho segment DATA và chỉ chứa duy nhất chỉ dẫn BYTE. Chỉ dẫn này cũng không có bất cứ tham số nào kèm theo.

- Định dạng: **.DSEG**

• **Chỉ dẫn ESEG:** dùng để bắt đầu cho segment EEPROM và cũng không kèm theo tham số nào.

- Định dạng: **.ESEG**

- **Chỉ dẫn ORG:** đặt vị trí bộ đếm chương trình với một giá trị địa chỉ tuyệt đối. Tùy thuộc vào chỉ dẫn ORG được đặt trong segment nào thì sẽ tương ứng với vị trí bộ đếm của segment đó.

- Định dạng: **.ORG** Biểu thức.

Ví dụ 4.9:

```

.DSEG          ;khai báo segment DATA
.ORG 0x120    ;đặt địa chỉ SRAM bắt đầu ở $120
VAR1: .BYTE 1  ;dành 1byte trong SRAM cho biến VAR1 tại địa chỉ $120
.CSEG          ;khai báo segment CODE
.ORG 0x10      ;đặt vị trí PC là $10
MOV R0,R1      ;
...            ; thực hiện 1 số lệnh nào đó
.....         ;
.ESEG          ;bắt đầu segment EEPROM
VAR2: .BYTE 5  ;dành 5 byte trong EEPROM cho biến VAR2

```

- *Lưu ý:* Giá trị mặc định cho bộ đếm chương trình của segment CODE và EEPROM là 0, còn segment DATA là địa chỉ theo ngay sau vùng I/O(cụ thể là 0x60 đối với các AVR không có I/O mở rộng, 0x100 hoặc cao hơn với những chip có I/O mở rộng).

- **Chỉ dẫn EQU:** gán một giá trị hằng số cho một tên ký hiệu.

- Định dạng: **.EQU** Ký hiệu = Biểu thức.

- **Chỉ dẫn SET:** cũng định nghĩa một giá trị hằng số hoặc một địa chỉ cố định, nhưng khác với chỉ dẫn EQU là chỉ dẫn SET có thể được gán lại bởi một giá trị khác sau đó.

- Định dạng: **.SET** Ký hiệu = Biểu thức.

Ví dụ 4.10:

```

.EQU COUNTER = 0xA0
.EQU MYADDR1=$08
.SET MYADDR2 = 0x18
LDI R26,MYADDR1 ;R26=$08
LDI R27,MYADDR2 ;R27=0x18
LDI R16,COUNTER ;R16 = 0xA0
.SET MYADDR2=0X0B ;định nghĩa lại MYADDR2=$0B
OUT MYADDR1,R16  ;MYADDR1=PORTC=0xA0
OUT MYADDR2, R16 ;MYADDR2=PORTD=0xA0

```

Trong ví dụ trên dòng thứ 4 và 5 MYADDR1 và MYADDR2 là dữ liệu. Dòng 7 định nghĩa lại giá trị ký hiệu MYADDR2=\$0B, thực hiện được điều này vì ban đầu dùng chỉ dẫn SET, nếu dùng chỉ dẫn EQU ban đầu trình biên dịch sẽ báo lỗi! Hai dòng cuối MYADDR1 và MYADDR2 là địa chỉ I/O, chính là PORTC và PORTD.

- **Chỉ dẫn DB:** dùng định nghĩa một hoặc nhiều byte dữ liệu cố định trong bộ nhớ chương trình (Flash ROM) hoặc EEPROM. Dữ liệu có thể là số thập phân, nhị phân, số thập lục phân hoặc mã ASCII.

- Định dạng: [nhãn:] **.DB** biểu thức[,biểu thức][,...]

Ví dụ 4.11:

```

.ORG $100
DATA1: .DB 1,8,5,3      ;mảng dữ liệu
DATA2: .DB 28             ;DECIMAL
DATA3: .DB 0b00110101    ;BINARY
DATA4: .DB 0x39           ;HEX
.ORG 0x110
DATA5: .DB 'Y'            ;ASCII
DATA6: .DB '2','0','0','5' ;ASCII
.ORG $116
DATA7: .DB "Hello"        ;chuỗi ASCII

```

- Lưu ý: trường hợp mảng dữ liệu có số byte lẻ, trình biên dịch sẽ lưu byte lẻ cuối cùng vào byte thấp địa chỉ ô nhớ cuối, bỏ qua byte cao và đưa ra dòng cảnh báo:
“**.cseg .db misalignment-padding zero byte**”
- **Chỉ dẫn DW:** tương tự chỉ dẫn DB nhưng dữ liệu là word(2byte) và không dùng cho chuỗi.
 - Định dạng: [nhãn:] .DW biểu thức[,biểu thức][,...]
- **Chỉ dẫn DD:** giống như chỉ dẫn DW nhưng dữ liệu là 32bit(2 word).
 - Định dạng: [nhãn:] .DD biểu thức[,biểu thức][,...]
- **Chỉ dẫn DQ:** tương tự như chỉ dẫn DW nhưng dữ liệu là 64bit(4 word).
 - Định dạng: [nhãn:] .DQ biểu thức[,biểu thức][,...]

Ví dụ 4.13:

```
.CSEG
DATA1: .DW 0x1234,0x1122
.ESEG
DATA2: .DD 0, 0xfadecabe, -2147483648, 1 << 30
DATA3: .DQ 0,0xfadecabedeadbeef, 1 << 62
```

- **Chỉ dẫn DEF:** dùng để định nghĩa cho ký hiệu là một thanh ghi.
 - Định dạng: .DEF Ký hiệu = thanh ghi

Ví dụ 4.14:

```
.ORG 0
.EQU NUM1=$F
.EQU NUM2=50
.DEF SUM = R22 ;định nghĩa ký hiệu SUM là t.ghi R22
    LDI R16,NUM1 ;
    LDI R17,NUM2 ;
    ADD R17,R16 ;R17= $F+50
    MOV SUM,R17 ;cất kết quả vào R22
HERE:
    JMP HERE ;dừng tại chỗ
```

- **Chỉ dẫn MESSAGE:** dùng để xuất 1 chuỗi, được sử dụng rộng rãi trong các điều khiển biên dịch.
 - Định dạng: .MESSAGE “<chuỗi>”
- **Chỉ dẫn ENDIF:** dùng để kết thúc điều khiển biên dịch.
 - Định dạng: .ENDIF
- **Chỉ dẫn IF:** theo sau chỉ dẫn IF là 1 biểu thức hằng. Nếu biểu thức này khác 0 thì đoạn lệnh theo sau sẽ được thực hiện, ngược lại sẽ thực hiện các đoạn lệnh theo sau chỉ dẫn ELSE hoặc ELIF(nếu có).
 - Định dạng: .IF biểu thức
- **Chỉ dẫn IFDEF, IFNDEF:** theo sau chỉ dẫn IFDEF/IFNDEF là một ký hiệu. Ký hiệu phải được định nghĩa bởi các chỉ dẫn EQU hoặc SET và không được dùng chỉ dẫn DEF cho ký hiệu này.
 - Định dạng: .IFDEF ký hiệu
 - .IFNDEF ký hiệu

Các lệnh theo sau chỉ dẫn IFDEF và trước chỉ dẫn ELSE hoặc ENDIF sẽ được thực hiện nếu ký hiệu theo sau chỉ dẫn IFDEF được định nghĩa. Ngược lại, nếu có 1 chỉ dẫn ELSE thì mọi lệnh từ chỉ dẫn ELSE đến ENDIF sẽ được thực hiện nếu ký hiệu chưa được định nghĩa.

Chỉ dẫn IFNDEF hoạt động tương tự chỉ dẫn IFDEF, nhưng thay trạng thái được định nghĩa thành chưa định nghĩa và ngược lại.

- **Chỉ dẫn ELIF:** chỉ dẫn ELIF thực hiện đoạn lệnh theo sau cho đến khi gặp chỉ dẫn ENDIF của chỉ dẫn ELIF cùng 1 cấp độ, nếu biểu thức theo sau chỉ dẫn ELIF đúng và cả mệnh đề IF ban đầu với các mệnh đề ELIF theo sau đều sai.

- Định dạng: .ELIF biểu thức

- **Chỉ dẫn ELSE:** chỉ dẫn ELSE thực hiện đoạn lệnh theo sau cho đến khi gặp chỉ dẫn ENDIF, nếu mệnh đề IF ban đầu và tất cả các ELIF đều sai.

- Định dạng: .ELSE

Ví dụ 4.15:

```
.IFDEF DEBUG
.MESSAGE "Debugging.." ;chỉ dẫn này sẽ được thực hiện nếu ký hiệu DEBUG được định nghĩa
.ELSE
.MESSAGE "Release.." ;ngược lại sẽ thực hiện chỉ dẫn này
.ENDIF
```

4.5 MACRO

- Macro được sử dụng như là 1 lệnh để thay thế cho 1 đoạn chương trình sau khi định nghĩa.
- Macro có thể đặt bất kỳ tại điểm nào trong chương trình.
- Định dạng: **.MACRO** tên macro
thân macro
.ENDMACRO
- Tên macro xem như là 1 lệnh(tùy gọi nhở) sau khi định nghĩa macro.
- Sau khi định nghĩa macro mới thực hiện được lệnh gọi macro theo tên của nó và các giá trị thích hợp được thay thế cho các tham số.
- Một macro có thể lên đến 10 tham số, các tham số được ký hiệu từ @0 đến @9.
- Sau khi biên dịch chương trình, tên macro được thay thế bằng toàn bộ nội dung trong thân macro theo đúng từng ký tự một.
- Việc thực thi lệnh macro có phần giống như lệnh gọi chương trình con, nhưng khác chương trình con ở chỗ không cần chuyển điều khiển cắt/phục hồi địa chỉ PC vào/ra stack.

Ví dụ 4.13: macro **LOADIO** sau thực hiện việc nạp một giá trị vào 1 thanh ghi I/O. Macro này được truyền bởi 2 tham số @0 là tên thanh ghi và @1 là giá trị cần nạp.

```
.MACRO LOADIO
```

```
    LDI R20,@1
    OUT @0,R20
```

```
.ENDMACRO
```

Đoạn chương trình sau thực hiện việc sử dụng macro trên để nạp giá trị vào các thanh ghi PORTA, DDRC và SPL.

```
LOADIO PORTA, 0x20 ;PORTA=$20
.EQU VAL_1 = 0xFF
LOADIO DDRC, VAL_1 ;DDRC=$FF
LOADIO SPL, 0x55 ;SPL=$55
```

Ví dụ 4.14: chương trình sau thực hiện lặp lại việc đảo các bit xen kẽ ở PORTB bằng cách sử dụng các macro LOADIO và DELAY

```
.MACRO LOADIO
```

```
    LDI R20,@1
    OUT @0,R20
```

```
.ENDMACRO
```

```
.MACRO DELAY
```

```
    LDI @0,@1
```

BACK:

```
    NOP
```

```
    NOP
```

```
    NOP
```

```
    NOP
```

```
    DEC @0
```

```
    BRNE BACK
```

```
.ENDMACRO
```

```
.ORG 0
```

```
L1: LOADIO DDRB,0xFF ;đặt PORTB ở chế độ xuất
    LOADIO PORTB,0x55 ;PortB = 0x55
    DELAY R18,0x70 ;tạo trễ
```

```

LOADIO PORTB,0xAA    ;PortB = 0xAA
DELAY   R18,0x70       ;tạo trễ
RJMP    L1

```

4.6 Hoạt động biên dịch hợp ngữ:

Có nhiều trình biên dịch hợp ngữ(assembler) cho AVR. Trong giáo trình này sẽ đề cập đến phần mềm **ATMEL STUDIO7**, có đầy đủ chức năng soạn thảo, biên dịch và mô phỏng, nó là một phần mềm phát triển hỗ trợ tất cả các chip AVR miễn phí.

- Trình biên dịch hợp ngữ(assembler) ATMEL STUDIO7 biên dịch thành các file sau:
 - File.OBJ: dùng để làm đầu vào cho trình mô phỏng hoặc trình giả lập.
 - File.LSS: chứa mã máy, mã nguồn, liệt kê các thanh ghi, các lệnh và dung lượng bộ nhớ được sử dụng trong chương trình.
 - File.MAP: liệt kê tất cả các nhãn được định nghĩa trong chương trình và giá trị tương ứng của nó.
 - File.HEX dùng để ghi vào ROM chương trình của AVR.

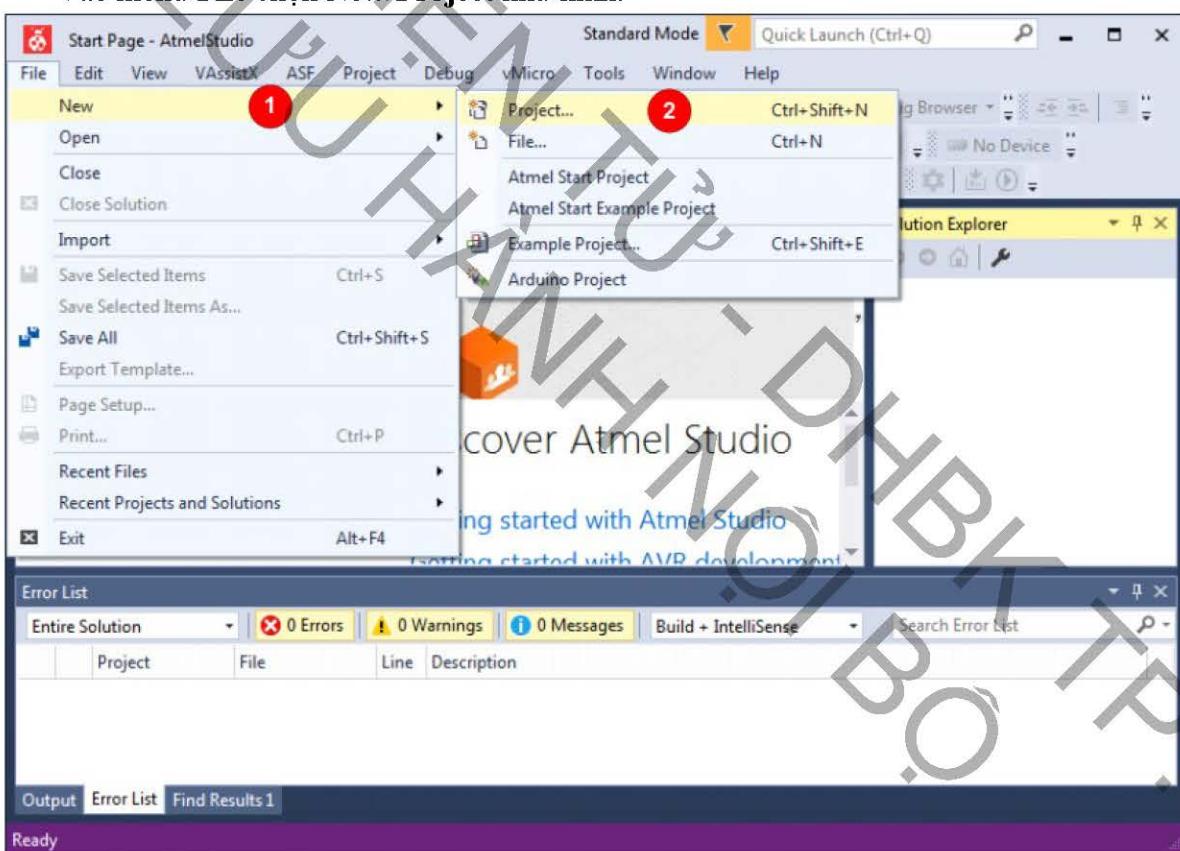
Tóm tắt các bước soạn thảo và biên dịch chương trình hợp ngữ:

1. Tài và cài đặt phần mềm ATMEL STUDIO7 từ website:

<http://www.microchip.com/avr-support/atmel-studio-7>

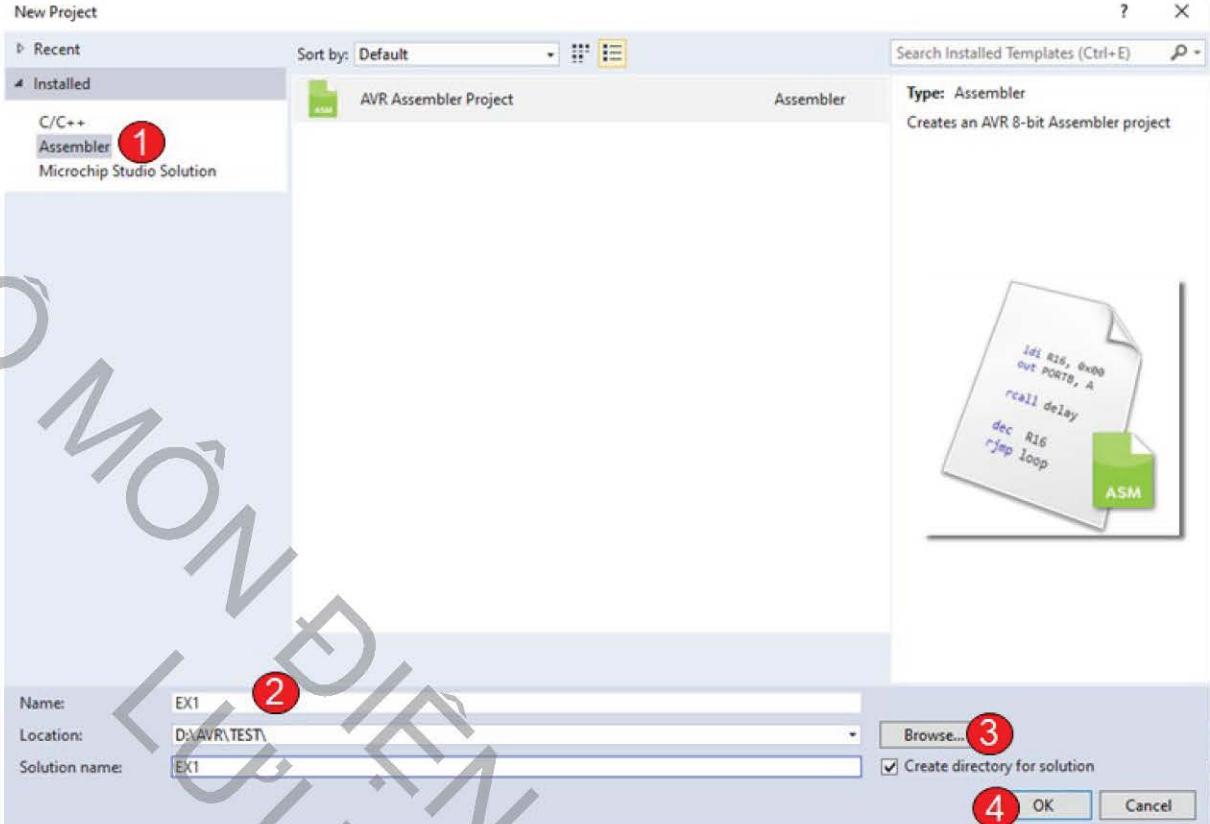
2. Chạy chương trình và tạo Project:

- Vào menu **File** chọn **New/Project** như hình.



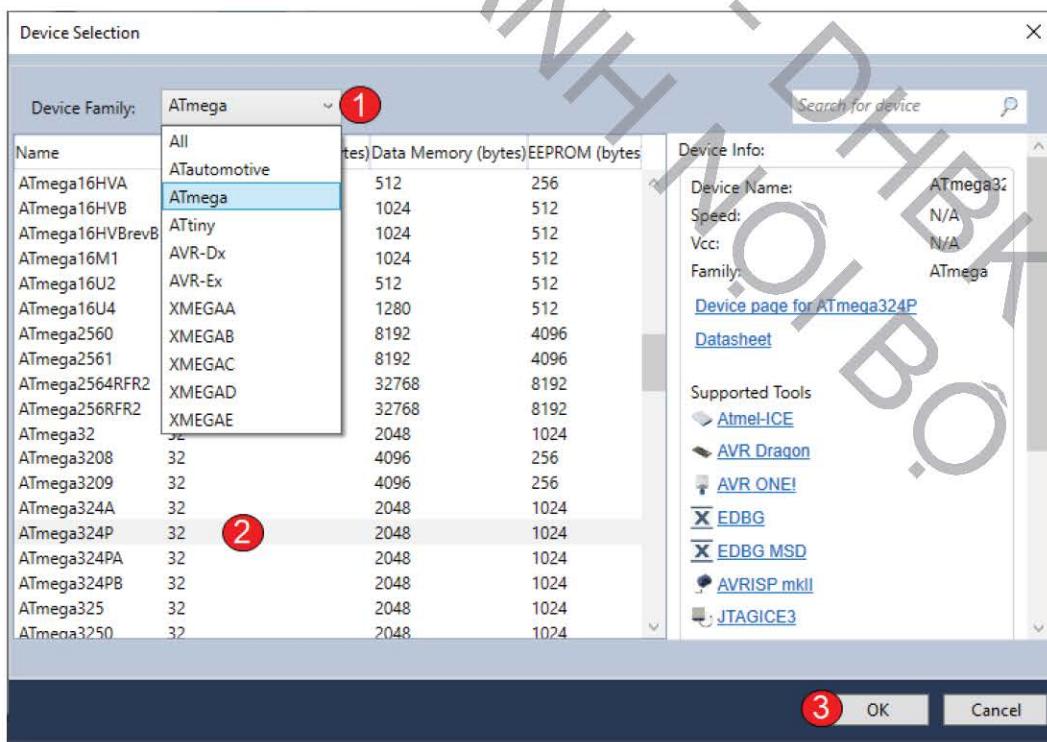
2. Trong hộp thoại được mở ra lần lượt thực hiện các bước theo thứ tự:

- a. Chọn **Assembler**.
- b. Đặt tên Project: ví dụ là **EX1**
- c. Chọn đường dẫn để lưu project bằng cách nhấp vào vị trí **Browse**.
- d. Sau đó nhấn **OK**.

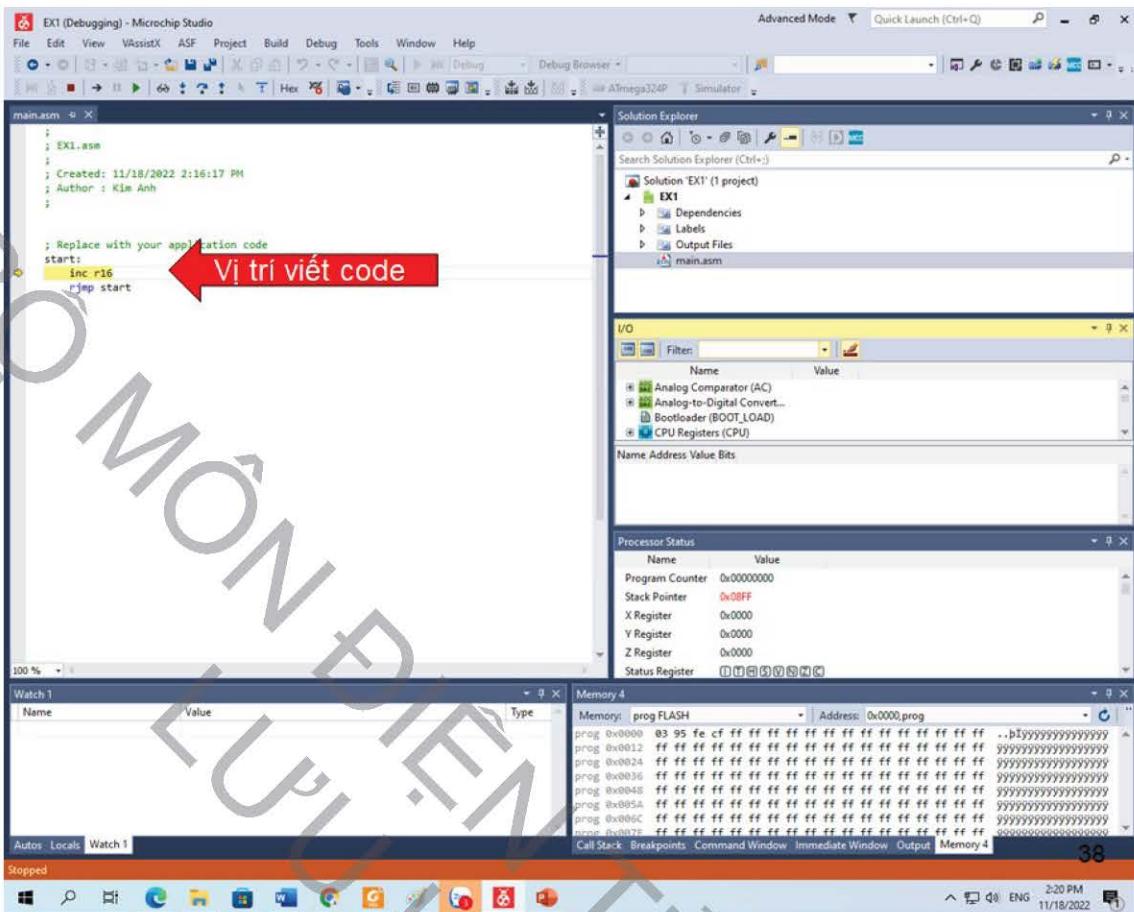


3. Trong hộp thoại Device Selection:

- Chọn ATmega cho Device Family.
- Chọn ATmega324P(hoặc chọn bất kỳ loại chip tương ứng được sử dụng).
- Sau đó nhấn OK.



Trình biên dịch sẽ tự động tạo ra một Project với tên EX1, và thêm 1 cửa sổ main.asm để người lập trình thực hiện việc soạn thảo chương trình.



Ví dụ sau thực hiện việc liên tục đảo trạng thái các bit ở PortB của vi điều khiển Atmega324P.
.ORG 0

```

LDI    R16,HIGH(RAMEND)
OUT   SPH,R16
LDI    R16,LOW(RAMEND)
OUT   SPL,R16
LDI    R16,0xFF
OUT   DDRB,R16

```

BACK:

```

COM   R16
OUT   PORTB,R16
RCALL DELAY
RJMP  BACK

```

DELAY:

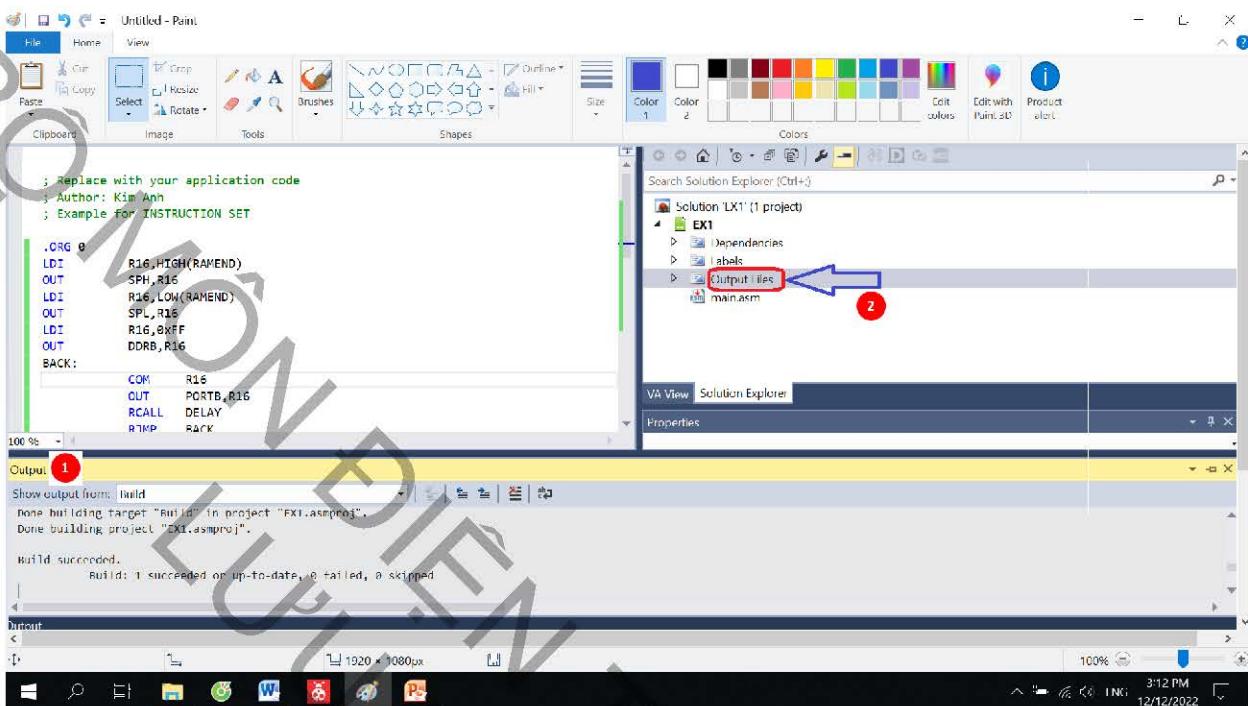
```

LDI   R20,16
L1:  LDI   R21,200
L2:  LDI   R22,250
L3:  NOP
     NOP
     DEC   R22
     BRNE L3
     DEC   R21
     BRNE L2
     DEC   R20
     BRNE L1
     RET

```

Gõ chương trình trên vào cửa sổ main.asm và thực hiện biên dịch chương trình.

4. Để biên dịch chương trình: nhấn F7 hoặc chọn **Build Solution** từ menu **Build**. Kết quả biên dịch quan sát ở cửa sổ **Output(1)**. Để quan sát nội dung các file được tạo ra, nhấn chuột vào vị trí **Output files(2)** trong cửa sổ **Solution Explorer** và chọn file tương ứng để quan sát.



PHÂN TÍCH ĐỊNH DẠNG NỘI DUNG CỦA FILE HEX ĐƯỢC TẠO RA

Thiết bị lập trình sử dụng file hex để nạp mã máy của lệnh hoặc dữ liệu vào Flash ROM, file hex phải có các thông tin bao gồm: các thông tin cần nạp, các thông tin tự tạo ra(kiểm tra tổng-checksum) và địa chỉ bắt đầu để nạp. Một cách tổng quát, mỗi dòng của file hex bao gồm 6 phần thông tin sau:

:BBAAAATTHHHH....HHHHCC

1. Mỗi dòng được bắt đầu bằng dấu hai chấm ":".
2. BB là số byte thông tin được nạp trên 1 dòng.
3. AAAA là địa chỉ để ghi(16bit-trong trường hợp dung lượng nhỏ hơn 64KB).
4. TT là đặc tính của dòng tương ứng. Có 3 giá trị 00, 01 và 02. Nếu là 00, thì sẽ còn nhiều dòng ở sau dòng hiện tại. 01 thì đó là dòng cuối và 02 chính là địa chỉ của segment hiện tại.
5. HHHH....HHHH là thông tin được ghi vào Flash(dữ liệu hoặc chương trình).
6. CC là một byte checksum.

Nguyên tắc tính byte checksum:

- Cộng tất cả các byte thông tin cần nạp và bỏ qua số nhớ.
- Lấy bù 2 của tổng này, đây chính là byte checksum và là byte cuối cùng trên 1 dòng của file hex.

Chương trình trên tạo ra file hex có nội dung sau:

```
:020000020000FC
:1000000008E00EBF0FEF0DBF0FEF04B9009505B963
:1000100001D0FCCF40E158EC6AEF000000006A9587
:0C002000E1F75A95C9F74A95B1F7089529
:00000001FF
```

Có thể tách các phần thông tin tương ứng như sau:

BB	AAAA	TT	HHHH....HHHH	CC
:02	0000	02	0000	FC
:10	0000	00	08E00EBF0FEF0DBF0FEF04B9009505B9	63
:10	0010	00	01D0FCCF40E158EC6AEF000000006A95	87
:0C	0020	00	E1F75A95C9F74A95B1F70895	29
:00	0000	01		FF

Nội dung của file lss liệt kê vị trí ô nhớ và nội dung là mã máy của các lệnh tương ứng.

LOC	OBJ	LINE
000000	e008	.ORG 0
000001	bf0e	LDI R16, HIGH(RAMEND)
000002	ef0f	OUT SPH, R16
000003	bf0d	LDI R16, LOW(RAMEND)
000004	ef0f	OUT SPL, R16
000005	b904	LDI R16, 0xFF
		OUT DDRB, R16
		BACK:
000006	9500	COM R16
000007	b905	OUT PORTB, R16
000008	d001	RCALL DELAY
000009	cffc	RJMP BACK
		DELAY:
00000a	e140	LDI R20, 16
00000b	ec58	L1: LDI R21, 200
00000c	ef6a	L2: LDI R22, 250
00000d	0000	L3: NOP
00000e	0000	NOP
00000f	956a	DEC R22
000010	f7e1	BRNE L3
000011	955a	DEC R21
000012	f7c9	BRNE L2
000013	954a	DEC R20
000014	f7b1	BRNE L1
000015	9508	RET

Để hiểu chi tiết hơn phần biên dịch và mô phỏng trong môi trường Atmel Studio 7, người đọc tham khảo thêm trong phần Phụ Lục 8.

CÂU HỎI ÔN TẬP

1. Xác định giá trị nạp vào các thanh ghi trong đoạn chương trình sau:

```
.EQU CONST1=0x10  
.EQU CONST2=0x91  
.EQU CONST3=0x14  
.EQU ADDR=(0x91 <<1)+1  
LDI R20,CONST1&CONST2  
LDI R21,CONST2|CONST3  
LDI R30,LOW(ADDR)  
LDI R31,HIGH(ADDR)
```

2. Xác định giá trị các thanh ghi R30,R31 và R20 khi thực thi đoạn chương trình sau

```
.ORG 0x0  
.EQU DATA_ADDR=(OUR_DATA<<1)  
LDI R30,LOW(DATA_ADDR)  
LDI R31,HIGH(DATA_ADDR)  
.ORG 0x100  
OUT_DATA: .DB 20,'A','C'
```

3. So sánh sự khác nhau giữa MACRO và chương trình con?

4. Tính byte checksum cho chuỗi sau: “Hello”.

5. Phân tích dữ liệu trong file hex ở mỗi trường hợp sau xem có bị lỗi hay không? Giải thích?

a. Data=\$65,\$09 và \$95; checksum=\$23.

b. Data=\$71,\$69, \$38 và \$81; checksum=\$6D.