Câu chuyện bắt đầu từ một cậu bé,
và một ý tưởng
có thể
làm thay đổi thế giới...

PAY IT FORWARD

Đó là khi bạn giúp đỡ 3 người bạn không quen biết,
dù là bằng thời gian,
hay công sức,
hay kinh nghiệm,
hay kiến thức,
hay tiền bạc, ...
của mình.

Mà không chờ đợi một sự báo ân nào.

Chỉ cần mỗi người trong 3 người đó,
lại đem những gì mình có, mà người khác cần,
tiếp tục giúp đỡ thêm 3 người nữa.

Chính những người-giúp-đỡ, và người-được-giúp-đỡ,
sẽ là những người góp phần thay đổi thế giới...

Một thế giới sẻ chia kiến thức - và yêu thương ...

PAY IT FORWARD ...

Chúng tôi không sáng tạo ra câu nói này.

Pay it forward...

Hãy tri ân người giúp mình bằng cách giúp đỡ người khác
Cho đi không phải để nhận lại.

C18 TRAINING

2 March 2019

payitforward.edu.vn


PAY IT FORWARD

1 About CPU

- **C**entral **P**rocessing **U**nit (CPU).
- **C**omplex **I**nstruction **S**et **C**omputer (CISC) vs **R**educed **I**nstruction **S**et **C**omputer (RISC) Approach.
- **Von-Neumann** vs **Harvard** architecture.

1 About CPU

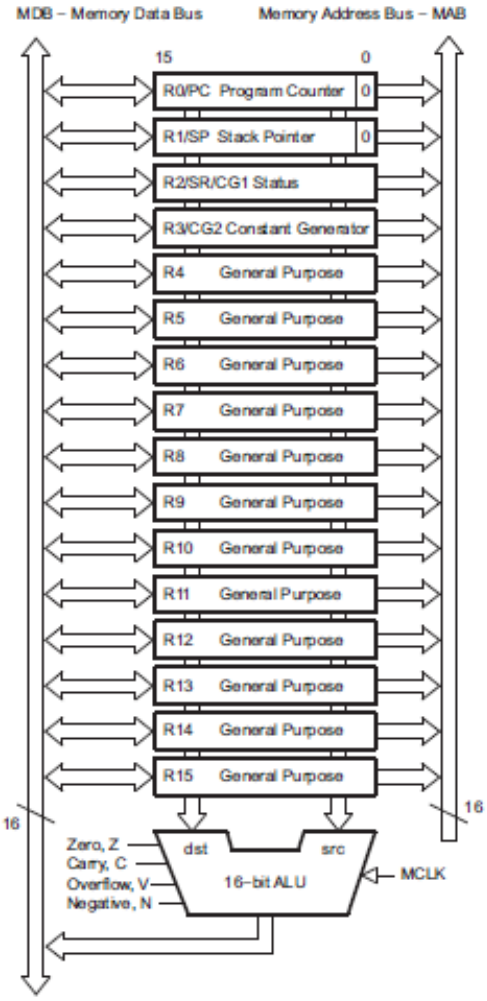
- CISC vs RISC Approach: *primary goal*
  - CISC: complete a task in as few lines of assembly as possible
  - RISC: simple instructions that can be executed within one clock cycle.

1 About CPU

- **Von-Neumann** vs **Harvard** architecture:
  - The Harvard architecture is a computer architechture with physically seperate storage and signal pathways for instructions and data.
  - The Von Neumann machines have shared signals and memory for code and data.

1 About CPU

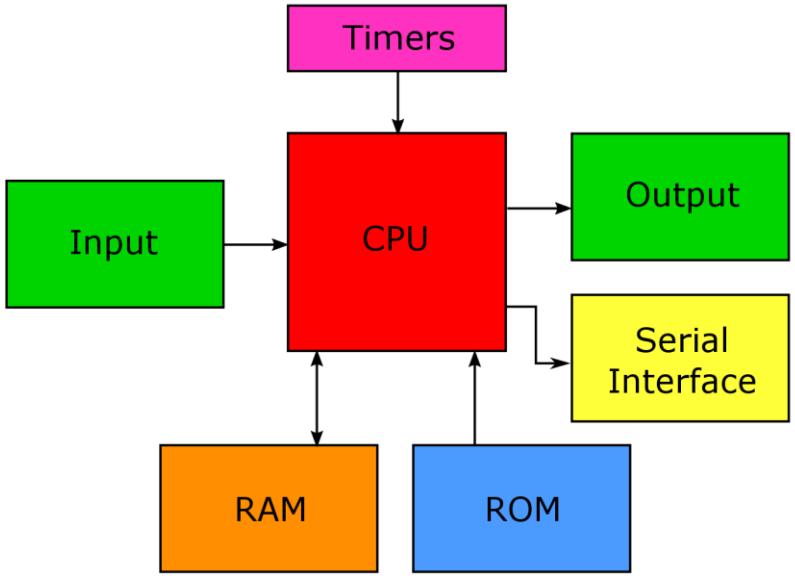
- CPU register
  - Program Couter (PC): point to next instruction to be executed.
  - Stack pointer (SP): store the return addresses of subroutine call and interrupts.
  - Status register (SR): show the status.
  - General Purpose: use as data registers, address pointers, or index values.



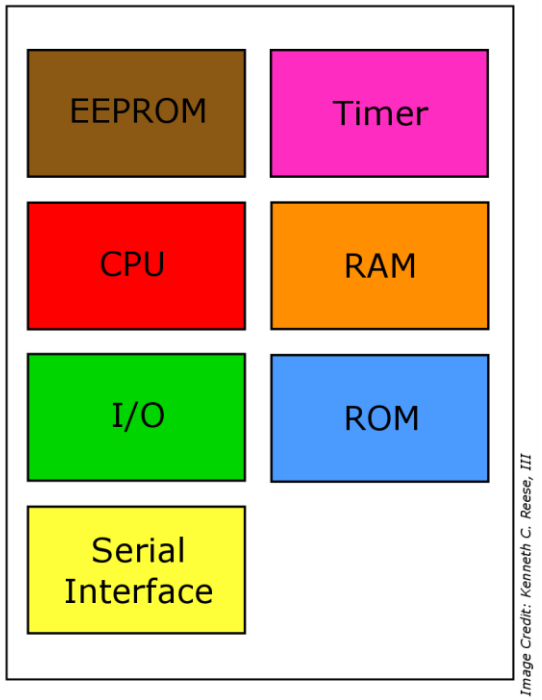
2 About MCU

- **M**icro**C**ontrollers **U**nit (MCU).
- MCU vs CPU:

Microprocesser: CPU
and several supporting chips.

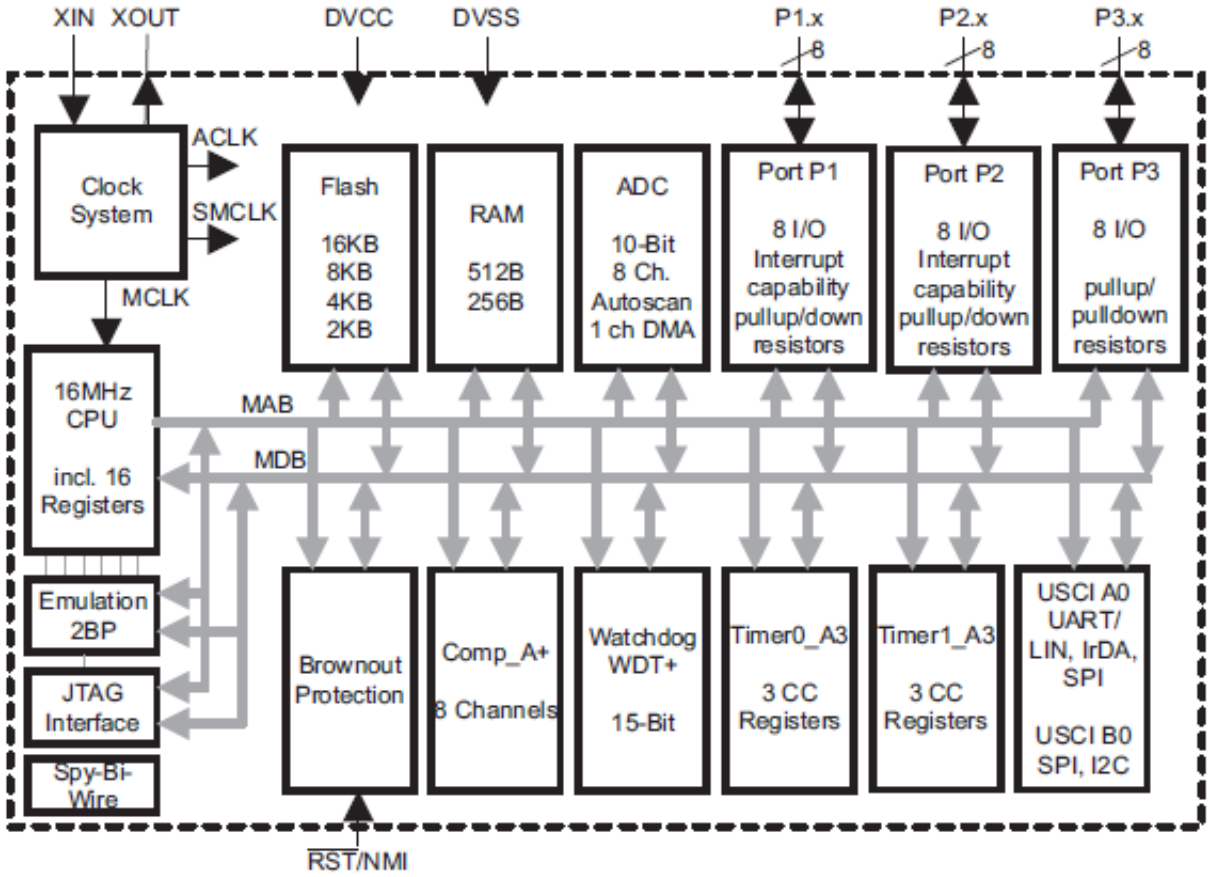


Microcontroller: CPU
on a single chip.



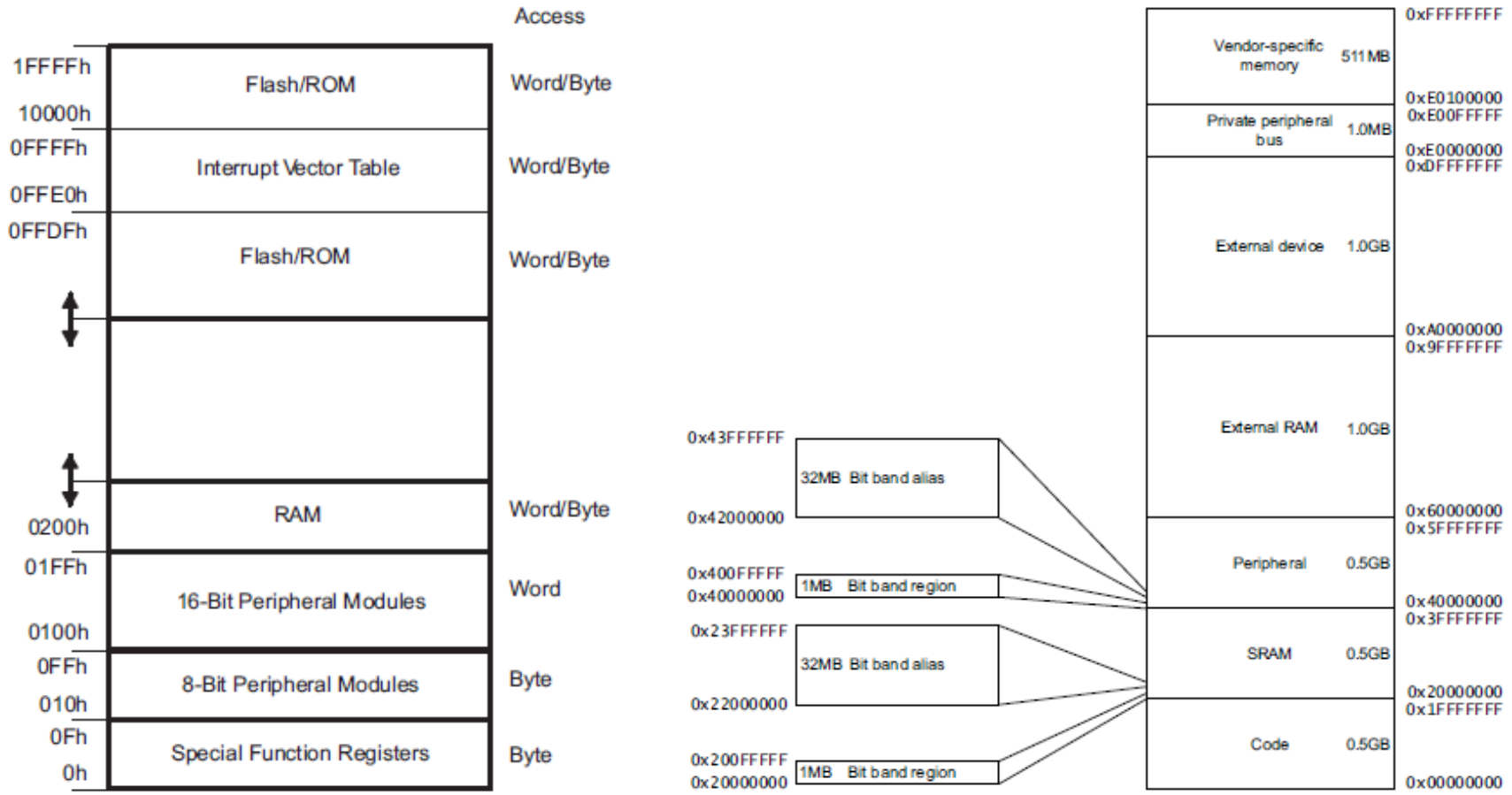
2 About MCU

- Example:
  - Functional Block Diagram MSP430G2xx5



2 About MCU

- Memory map:
  - MSP430 and Cortex-M4



2 About MCU

- Memory map:
  - Code.
  - Flash/ROM.
  - RAM.
  - Peripheral modules.
  - Interrupt vector table.
  - ...

2 About MCU

- Memory map:
  - Interrupt vector table (MSP430G2553).

Table 5. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range(1)	PORIFG RSTIFG WDTIFG KEYV(2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG(2)(3)	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCIFG(4)	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG(2)(4)	maskable	0FFF8h	28
Comparator_A+	CAIFG(4)	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG(4)	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (5)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG(2)(5)	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG(2)(6)	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG(4)	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7(2)(4)	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7(2)(4)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See (7)			0FFDEh	15
See (8)			0FFDEh to 0FFC0h	14 to 0, lowest

2 About MCU

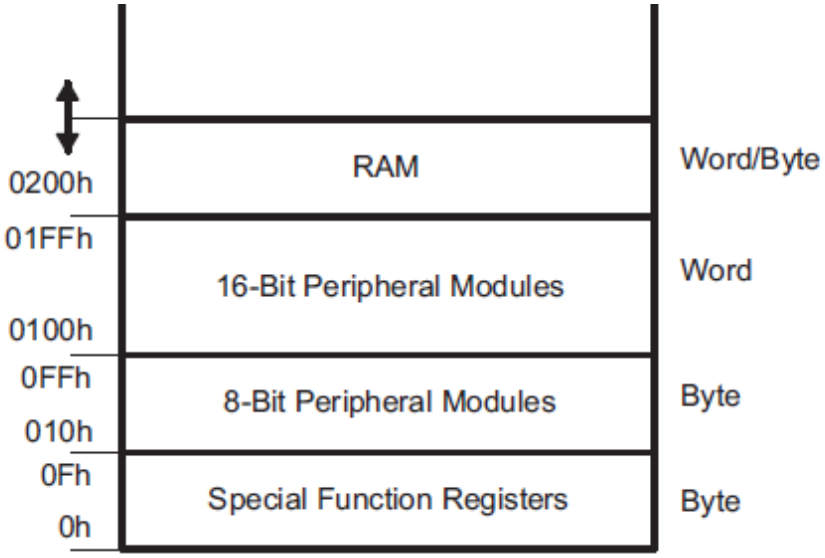
- Memory map:
  - Linker file: *lnk_msp430g2553.cmd*
    - Specify the system memory map.
    - Specify the sections allocation into memory.

```
MEMORY
{
    SFR                     : origin = 0x0000, length = 0x0010
    PERIPHERALS_8BIT        : origin = 0x0010, length = 0x00F0
    PERIPHERALS_16BIT       : origin = 0x0100, length = 0x0100
    RAM                     : origin = 0x0200, length = 0x0200
    INFOA                   : origin = 0x10C0, length = 0x0040
    INFOB                   : origin = 0x1080, length = 0x0040
    INFOC                   : origin = 0x1040, length = 0x0040
    INFOD                   : origin = 0x1000, length = 0x0040
    FLASH                   : origin = 0xC000, length = 0x3FDE
    BSLSIGNATURE            : origin = 0xFFDE, length = 0x0002, fill = 0xFFFF
    INT00                   : origin = 0xFFE0, length = 0x0002
    INT01                   : origin = 0xFFE2, length = 0x0002
    INT02                   : origin = 0xFFE4, length = 0x0002
    INT03                   : origin = 0xFFE6, length = 0x0002
    INT04                   : origin = 0xFFE8, length = 0x0002
    INT05                   : origin = 0xFFEA, length = 0x0002
    INT06                   : origin = 0xFFEC, length = 0x0002
    INT07                   : origin = 0xFFEE, length = 0x0002
    INT08                   : origin = 0xFFF0, length = 0x0002
    INT09                   : origin = 0xFFF2, length = 0x0002
    INT10                   : origin = 0xFFF4, length = 0x0002
    INT11                   : origin = 0xFFF6, length = 0x0002
    INT12                   : origin = 0xFFF8, length = 0x0002
    INT13                   : origin = 0xFFFA, length = 0x0002
    INT14                   : origin = 0xFFFC, length = 0x0002
    RESET                   : origin = 0xFFFE, length = 0x0002
}
```

2 About MCU

- Memory map:
  - Linker file: *lnk_msp430g2553.cmd*

```
PERIPHERALS_16BIT  : origin = 0x0100, length = 0x0100
RAM                : origin = 0x0200, length = 0x0200
```



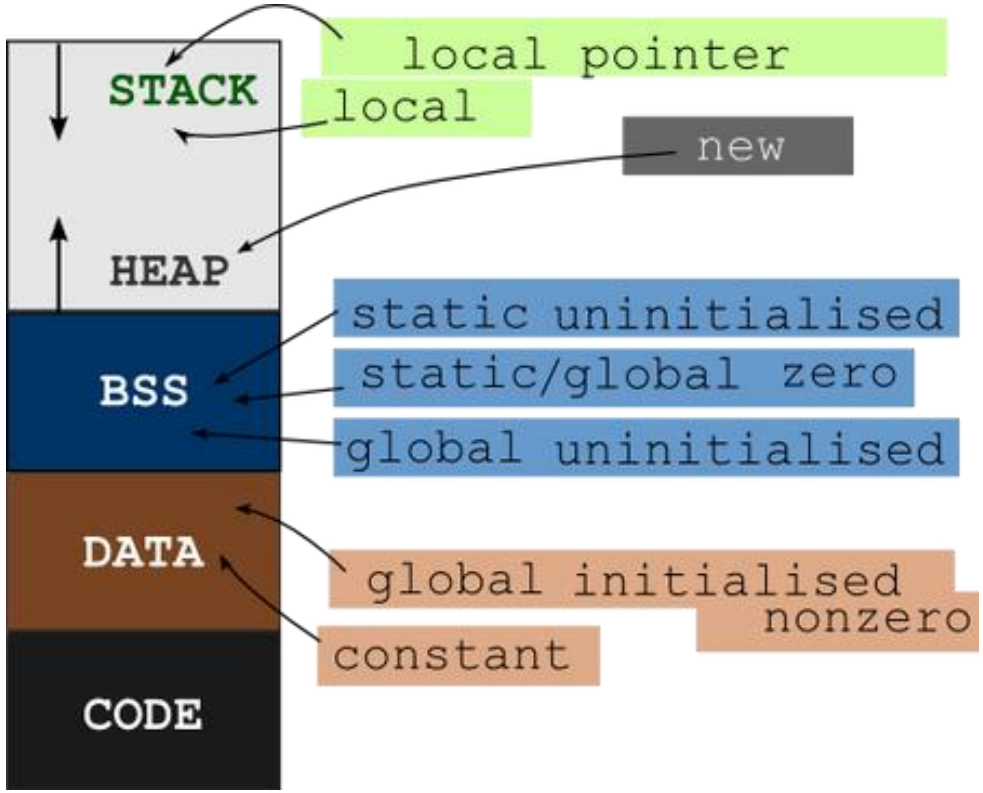
Device	BSL	EEM	Flash (KB)	RAM (B)
MSP430G2553IRHB32 MSP430G2553IPW28 MSP430G2553IPW20 MSP430G2553IN20	1	1	16	512

```
SECTIONS
{
    .bss         : {} > RAM                  /* Global & static vars
    .data        : {} > RAM                  /* Global & static vars
    .TI.noinit   : {} > RAM                  /* For #pragma noinit
    .sysmem      : {} > RAM                  /* Dynamic memory allocation area
    .stack       : {} > RAM (HIGH)           /* Software system stack

    .text        : {} > FLASH                /* Code
    .cinit       : {} > FLASH                /* Initialization tables
    .const       : {} > FLASH                /* Constant data
    .bslsignature  : {} > BSLSIGNATURE       /* BSL Signature
    .cio         : {} > RAM                  /* C I/O Buffer

    .pinit       : {} > FLASH                /* C++ Constructor tables
    .binit       : {} > FLASH                /* Boot-time Initialization tables
    .init_array  : {} > FLASH                /* C++ Constructor tables
    .mspabi.exidx : {} > FLASH               /* C++ Constructor tables
    .mspabi.extab : {} > FLASH               /* C++ Constructor tables
#ifdef __TI_COMPILER_VERSION__
  #if __TI_COMPILER_VERSION__ >= 15009000
    #ifndef __LARGE_DATA_MODEL__
    .TI.ramfunc : {} load=FLASH, run=RAM, table(BINIT)
    #else
    .TI.ramfunc : {} load=FLASH | FLASH2, run=RAM, table(BINIT)
    #endif
  #endif
#endif
```

```
    .infoA     : {} > INFOA              /* MSP430 INFO FLASH Memory segments *
    .infoB     : {} > INFOB
    .infoC     : {} > INFOC
    .infoD     : {} > INFOD

    /* MSP430 Interrupt vectors           */
    TRAPINT      : { * ( .int00 ) } > INT00 type = VECT_INIT
    .int01       : {}               > INT01
    PORT1        : { * ( .int02 ) } > INT02 type = VECT_INIT
    PORT2        : { * ( .int03 ) } > INT03 type = VECT_INIT
    .int04       : {}               > INT04
    ADC10        : { * ( .int05 ) } > INT05 type = VECT_INIT
    USCIAB0TX    : { * ( .int06 ) } > INT06 type = VECT_INIT
    USCIAB0RX    : { * ( .int07 ) } > INT07 type = VECT_INIT
    TIMER0_A1    : { * ( .int08 ) } > INT08 type = VECT_INIT
    TIMER0_A0    : { * ( .int09 ) } > INT09 type = VECT_INIT
    WDT          : { * ( .int10 ) } > INT10 type = VECT_INIT
    COMPARATORA   : { * ( .int11 ) } > INT11 type = VECT_INIT
    TIMER1_A1    : { * ( .int12 ) } > INT12 type = VECT_INIT
    TIMER1_A0    : { * ( .int13 ) } > INT13 type = VECT_INIT
    NMI          : { * ( .int14 ) } > INT14 type = VECT_INIT
    .reset       : {}               > RESET  /* MSP430 Reset vector         */
}
```

2 About MCU

- Memory map:
  - Linker file: *lnk_msp430g2553.cmd*
    - Memory Layout of C Program



2 About MCU

- Memory map:
  - Linker file: *lnk_msp430g2553.cmd*
    - `.text`: executable code.
    - `.const`: initialized global variables.
    - `.cinit`: tables which initialize global variables.
    - `.init_array` or `.pinit`: Table of C++ constructors called at startup.
    - `.binit`: boot-time Initialization tables.
    - `.data`: global & static vars (initialized data).
    - `.bss`: global & static vars (no initialized/initialized to zero).
    - `.sysmem`: malloc heap.
    - `.stack`: system stack.

2 About MCU

- Memory map:
  - Linker file: *lnk_msp430g2553.cmd*
    - Interrupt Vector Table (see in datasheet)

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-Up External Reset Watchdog Timer+ Flash key violation PC out-of-range(1)	PORIFG RSTIFG WDTIFG KEYV(2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG(2)(3)	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCIFG(4)	maskable	0FFFAh	29
Timer1_A3	TA1CCR2 TA1CCR1 CCIFG, TAIFG(2)(4)	maskable	0FFF8h	28
Comparator_A+	CAIFG(4)	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG(4)	maskable	0FFF2h	25
Timer0_A3	TA0CCR2 TA0CCR1 CCIFG, TAIFG (5)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 receive USCI_B0 I2C status	UCA0RXIFG, UCB0RXIFG(2)(5)	maskable	0FFEEh	23
USCI_A0/USCI_B0 transmit USCI_B0 I2C receive/transmit	UCA0TXIFG, UCB0TXIFG(2)(6)	maskable	0FFECh	22
ADC10 (MSP430G2x53 only)	ADC10IFG(4)	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (up to eight flags)	P2IFG.0 to P2IFG.7(2)(4)	maskable	0FFE6h	19
I/O Port P1 (up to eight flags)	P1IFG.0 to P1IFG.7(2)(4)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
See (7)			0FFDEh	15
See (8)			0FFDEh to 0FFC0h	14 to 0, lowest

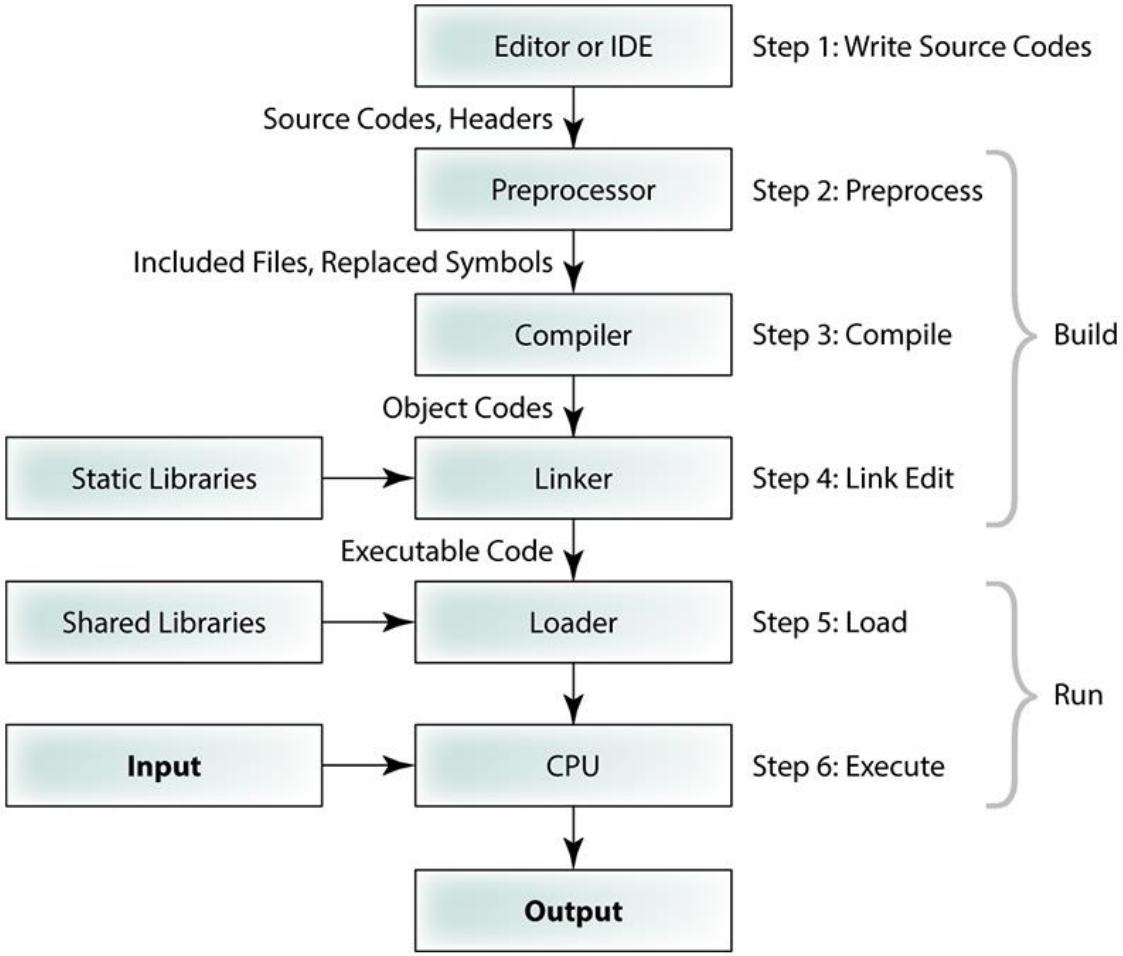
2 About MCU

- Memory map:
  - Linker address map (.map):
    - An output from the linker that provides a summary of the memory configuration, section allocation, addresses of external symbols after they have been relocated, and more.
    - Check the size of code and data sections in your program and where the sections and symbols are allocated in memory
    - Note:

```
OUTPUT FILE NAME:   <[MSP430] Sample Project.out>
ENTRY POINT SYMBOL: "_c_int00_noargs_noexit"  address: 0000c41c
```

2 About MCU

- Build process:



2 About MCU

- Build process:
  - **Preprocess**:
    - Removal of comment.
    - Expansion of macros.
    - Expansion of the included file.
    - An output file is **<file_name>.i.**
  - **Compile**:
    - Compile <file_name>.i into **<file_name>.s** (assembly level instructions).

2 About MCU

- Build process:
  - **Assembly**:
    - Assembler takes <file_name>.s and produces **<file_name>.o** (machine level instructions).
  - **Link**:
    - All the linking of function calls with their definitions are done.
    - The output file is an executable file (depends on hardware).

2 About MCU

- Build process (additional):
  - **Shared** libraries vs **Static** libraries**:**
    - **Shared libraries (.so** or **.dll)**: ***referenced*** by programs using it at ***run-time***.
    - **Static libraries** (**.a** or **.lib**): ***directly linked*** into the program at ***compile time***. A program using a static library takes copies of the code that it uses from the static library and makes it ***part of the program***.

3 About IDE

- **I**ntegrated **D**evelopment **E**nvironment.
- Consist:
  - Source code editor.
  - Compiler.
  - Code completion.
  - Debugger.

3 Coding style

- A set of **rules** or **guidelines** used when writing the source code for a computer program (code conventions).
- Referent thread (official members ONLY):
  - http://www.payitforward.edu.vn/forum/threads/2491/

4 C/C++ Optimization

- Target:
  - Run faster.
  - Use less memory or other source.
  - Save energy.
  - Reduce size of programs.

4 C/C++ Optimization

- Levels of optimization:
  - Design.
  - Algorithms and data structures.
  - **Source code**.
  - Compile.
  - Assemby.
  - Run time.

4 C/C++ Optimization

- Consider about optimization?
  - Less readable/maintainable.
  - More complex.
- Profiling tool:
  - Space (memory).
  - Time complexity of a program.
  - Usage of particular instructions.
  - Frequency and duration of function calls.
  - Ex: Windows (Visual Studio) and Linux (Perf, Gprof).

4 C/C++ Optimization

- The 5 steps to optimize:
  - Code must run correctly.
  - Use profiler to detect and diagnose slow code.
  - Optimize code.
  - Confirm that the optimized code run correctly.
  - Use profiler to verify the result.
- Unit: cycle.

4 C/C++ Optimization

- **Varible storage**: Static storage
  - Access to static varible is faster than to local variable.

```
int n = 100000000;
for(int i = 0; i < n; i++)
{
  <code>
}
```

```
static int n = 100000000;
for(int i = 0; i < n; i++)
{
  <code>
}
```

4 C/C++ Optimization

- **Varible storage**: Static storage
  - Replace calculation result by storing the value in static storage, where possible.

```
long result = 0;
for(int i = 0; i < 100000000; i++)
{
  int j = i % 10;
  result += j * j;
}
```

```
long result = 0;
static const long table[10] =
  {0, 1, 4, 9, 16, 25, 36, 49, 64, 81};

for(int i = 0; i < 100000000; i++)
{
  int j = i % 10;
  result += table[j];
}
```

4 C/C++ Optimization

- **Varible storage**: Register storage
  - Variables stored in register are accessed in a flash.
  - The number of registers is strictly limited.

```
static int x = 10;
long result = 0;
for(int i = 0; i < 100000000; i++)
{
  result += i * x;
}
```

```
register int x = 10;
long result = 0;
for(int i = 0; i < 100000000; i++)
{
  result += i * x;
}
```

4 C/C++ Optimization

- **Operators**: Initializations
  - Use initialzations instead of assignments. In particular, in constructors, use initialization lists.

```
for(int i = 0; i < 100000000; i++)
{
  string s;
  s = "PIF Lab";
}
```

```
for(int i = 0; i < 100000000; i++)
{
  string s("PIF Lab");
}
```

4 C/C++ Optimization

- **Operators**: Initializations
  - Use initialzations instead of assignments. In particular, in constructors, use initialization lists.

```
myClass(const T& t)
{
    mValue = t;
}
```

```
myClass(const T& t) : mValue(t){}
```

4 C/C++ Optimization

- **Operators**: Initializations
  - Use initialzations instead of assignments. In particular, in constructors, use initialization lists.

```
string s1("PIF");
string s2("Lab");

for(int i = 0; i < 100000000; i++)
{
  string s3 = s1 + " " + s2 "!";
}
```

```
string s1("PIF");
string s2("Lab");

for(int i = 0; i < 100000000; i++)
{
  string s3 = s1;
  s3 += " ";
  s3 += s2;
  s3 += "!";
}
```

4 C/C++ Optimization

- **Operators**: Initializations
  - In almost all classes, use the assignment composite operators instead of simple operators combined with assignment opeartors.

```
string s1("PIF");
string s2("Lab");

for(int i = 0; i < 100000000; i++)
{
  string s3 = s1 + " " + s2 "!";
}
```

```
string s1("PIF");
string s2("Lab");

for(int i = 0; i < 100000000; i++)
{
  string s3 = s1;
  s3 += " ";
  s3 += s2;
  s3 += "!";
}
```

4 C/C++ Optimization

- **Operators**: Other
  - For objects, use the prefix operator (++obj) instead of the postfix operator (obj++).
  - Use shift operations >> and << instead of integer multiplication and division if possible.
  - Should use same type of variables for processing, type conversion must be avoid.
  - Replace integer division with multiplication when there are multiple divisions in an expression.

```
x = i/j/k;
```

```
x = i/(j*k);
```

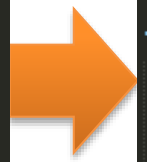
4 C/C++ Optimization

- **Booleans**:
  - If one operand is more predictable than the other, then put it first.
  - If one operand is faster to calculate than the other then put it first.
  - The one operand of the Boolean operators &&: the first operand of && is false, then the second operand is not evaluated at all because the result is known to be false regardless of the value of the second operand.

4 C/C++ Optimization

- **Functions**: Macro
  - Use macros instead of functions

```
int max(int a, int b)
{
    return (a > b)?a:b;
}

void main(void)
{
    for(int i = 0; i < 100000000; i++)
    {
        int x = max (i, 10000);
    }
}
```



```
#define MAX(a,b)        (a>b?a:b)

void main(void)
{
    for(int i = 0; i < 100000000; i++)
    {
        int x = MAX(i, 10000);
    }
}
```

4 C/C++ Optimization

- **Functions**: Parameters
  - Pass structures or classes by reference, not by value.

```
int rs = 0;
void foo(vector<int> a)
{
    for (int i = 0; i < a.size(); i++)
    {
        rs += a[i];
    }
}
```

```
int rs = 0;
void foo(vector<int> &a)
{
    for (int i = 0; i < a.size(); i++)
    {
        rs += a[i];
    }
}
```

4 C/C++ Optimization

- **Functions**: Other rules
  - Return objects via reference parameters.
  - The return value of a function will be stored in a register. If this return data has no intended usage, time and space are wasted in storing this information.

4 C/C++ Optimization

- **Array**:

```
array[i][j] + array[i][j+1]
```

```
array[i][j] + array[i+1][j]
```

4 C/C++ Optimization

- **Loops**: function calls
  - Move loops inside function calls.

```
int rs = 0;
void foo(void)
{
    rs++;
}

void main (void)
{
    for(int i = 0; i < 100000000; i++)
    {
        foo();
    }
}
```

```
int rs = 0;
void foo(void)
{
    for(int i = 0; i < 100000000; i++)
    {
        rs++;
    }
}

void main (void)
{
    foo();
}
```

4 C/C++ Optimization

- **Loops**: Other tip
  - Use the prefix operator (++i) instead of postfix operator (i++).
  - It is faster to test if something is equal to zero than to compare two different numbers.

```
for (int i = 0; i < 10; i++)
```

```
for (int i = 10; i--;)
```

  - Avoid calculation in loop condition.

```
for (int i = 0; i < a * b - 9 * c; i++)
```

PAY IT FORWARD

payitforward.edu.vn