Université de Savoie
Module I621

Travaux Dirigés

Architecture des Ordinateurs

Sylvain MONTAGNY
sylvain.montagny@univ-savoie.fr
Bâtiment chablais, bureau 13
04 79 75 86 86

TD1 : Rappels, Cadencement d'un microprocesseur
TD2 : Le pipeline
TD3 : Les mémoires caches
TD4 : Les interruptions
TD5 : Les accès DMA

Université de Savoie
Module I621
Architecture des ordinateurs

TD 1
Rappels sur les architectures à microprocesseurs

Un processeur 64 bits stocke les données binaires qu'il traite dans des circuits intégrés de mémoire RAM. La capacité de chaque circuit mémoire est de 4 Mo, les données binaires étant organisées en mots de 64 bits. La capacité totale de l'ensemble des mémoires vives est de 32 Mo.

Q1. De combien de circuits différents est constitué l'ensemble de la mémoire vive associée à ce processeur ?

Q2. Donner le nombre de cases mémoires disponibles dans chaque circuit RAM ainsi que le nombre total de cases mémoires pour l'ensemble des circuits.

Q3. Quelle doit être la taille minimum du bus d'adresse de ce processeur ? Quelle est l'adresse la plus haute et l'adresse la plus basse (en hexadécimal) accessible par le processeur ?

La taille du bus d'adresse sera désormais la taille minimum que vous venez de trouvez.

Q4. Parmi l'ensemble des bits constituant le bus adresse, donner le nombre de bits réservés à la sélection d'un circuit mémoire (boitier) et le nombre de bits réservés à la sélection d'un emplacement dans cette mémoire.

Q5. Donner les adresses de début et de fin des quatre premiers circuits en complétant le tableau ci-dessous.

Circuits	A_{21} A_{20}	A_{19} A_{16}	A_{15} A_{12}	A_{11} A_{8}	A_{7} A_{4}	A_{3} A_{0}	Adresses Hexadécimal	
RAM 3							Fin	
							Début	
RAM 2							Fin	
							Début	
RAM 1							Fin	
							Début	
RAM 0							Fin	
							Début	

Tableau 1 : Adressage des circuits RAM

Q6. Représentez sur un schéma l'ensemble des circuits, du processeur et des bus (on ne représentera que deux RAMs). On devra faire apparaitre clairement le nombre de fils sur chaque partie des bus.

Université de Savoie
Module I621 : Architecture des Ordinateurs

TD 2
Pipeline

I.Pipeline dans un PIC16F

La documentation des microcontrôleurs PIC16FXXXX montre la présence d'un pipeline (voir page suivante).
Dans le document Q_{cycle} signifie « clock cycle » (période d'horloge).

Q1. Quelles est le rapport en une période d'horloge (Q_{cycle}) et une « instruction cycle ».

Q2. Quelle est la profondeur du pipeline de ce microcontrôleur? Combien d' « instruction cycle » prend une instruction pour s'exécuter la plupart du temps.

Q3. Expliquer le « Flush » dans le pipeline, combien de cycle instructions aura mis l'instruction situé à l'adresse SUB_1 pour s'exécuter.

4.3 Instruction Flow/Pipelining

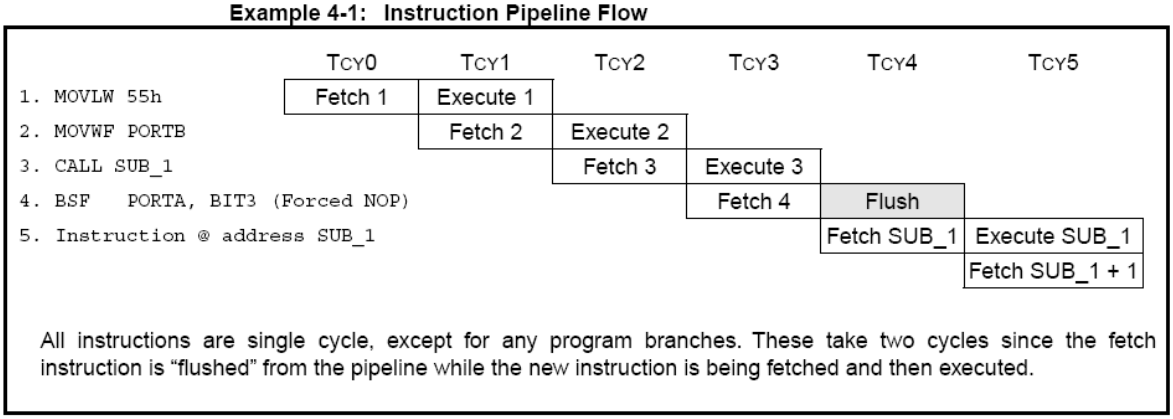
An "Instruction Cycle" consists of four Q cycles (Q1, Q2, Q3, and Q4). Fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to Pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g. GOTO) then an extra cycle is required to complete the instruction (Example 4-1).

The instruction **fetch** begins with the program counter incrementing in Q1.

In the **execution** cycle, the fetched instruction is latched into the "Instruction Register (IR)" in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

Example 4-1 shows the operation of the two stage pipeline for the instruction sequence shown. At time TCY0, the first instruction is fetched from program memory. During TCY1, the first instruction executes while the second instruction is fetched. During TCY2, the second instruction executes while the third instruction is fetched. During TCY3, the fourth instruction is fetched while the third instruction (CALL SUB_1) is executed. When the third instruction completes execution, the CPU forces the address of instruction four onto the Stack and then changes the Program Counter (PC) to the address of SUB_1. This means that the instruction that was fetched during TCY3 needs to be "flushed" from the pipeline. During TCY4, instruction four is flushed (executed as a NOP) and the instruction at address SUB_1 is fetched. Finally during TCY5, instruction five is executed and the instruction at address SUB_1 + 1 is fetched.

Example 4-1: Instruction Pipeline Flow



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

II.Pipeline dans un DSP TMS320C5416

Ci-dessous est représenté le schéma de l'organisation des bus internes du DSP.

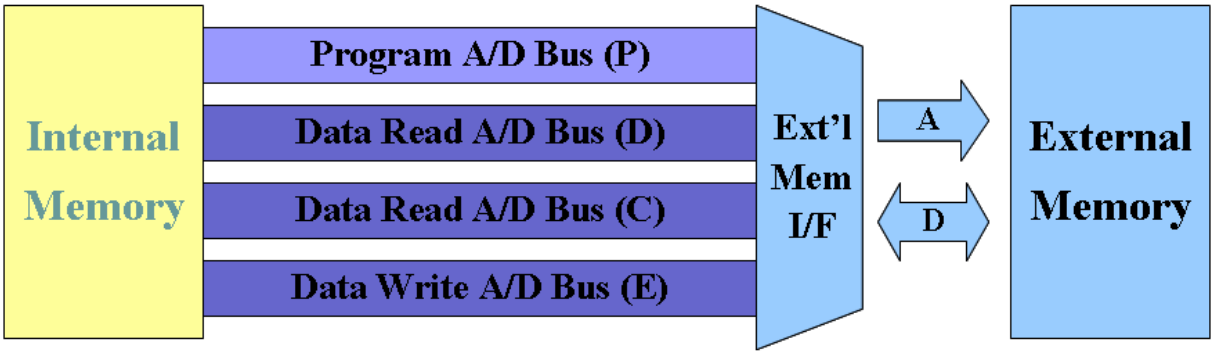
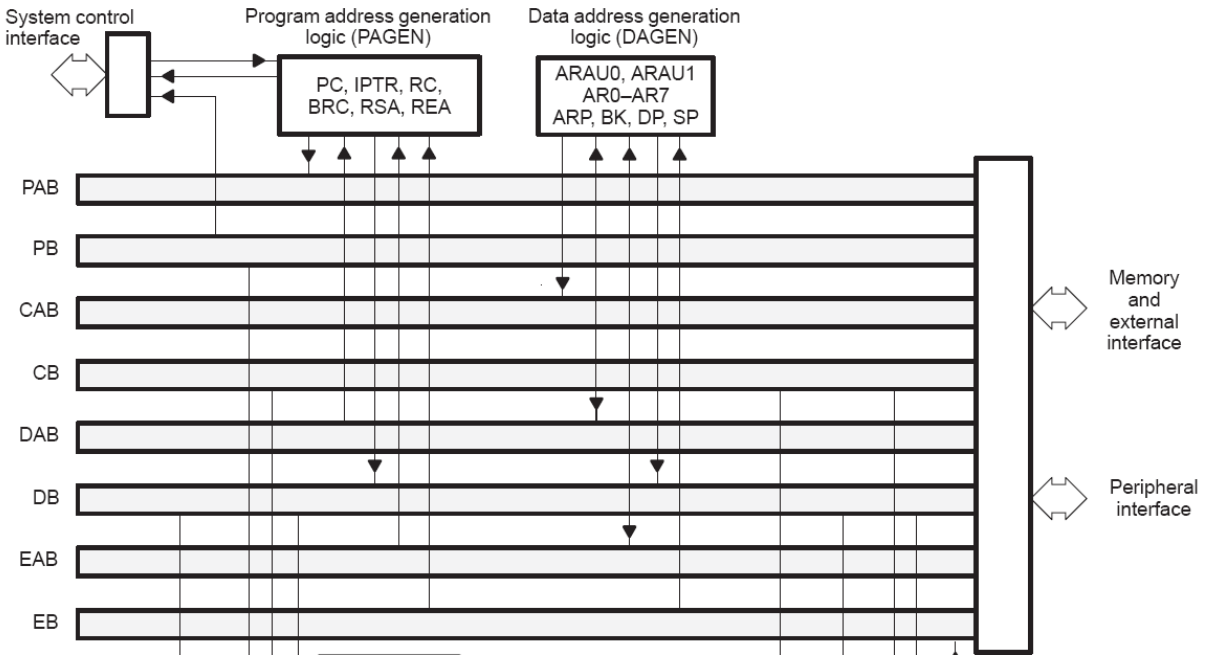


Figure 1 : Représentation simplifiée des bus du processeur

Q1. Combien de bus différents sont représentés sur le schéma précédent, et a quelles catégories de mémoires sont ils connectés.

Le document constructeur est en fait le suivant :

Figure 2–1. Block Diagram of TMS320C54x DSP Internal Hardware



2.1 Bus Structure

The C54x™ DSP architecture is built around eight major 16-bit buses (four program/data buses and four address buses):

- The program bus (PB) carries the instruction code and immediate operands from program memory.
- Three data buses (CB, DB, and EB) interconnect to various elements, such as the CPU, data address generation logic, program address generation logic, on-chip peripherals, and data memory.
  - The CB and DB carry the operands that are read from data memory.
  - The EB carries the data to be written to memory.
- Four address buses (PAB, CAB, DAB, and EAB) carry the addresses needed for instruction execution.

The C54x DSP can generate up to two data-memory addresses per cycle using the two auxiliary register arithmetic units (ARAU0 and ARAU1).

The PB can carry data operands stored in program space (for instance, a coefficient table) to the multiplier and adder for multiply/accumulate operations or to a destination in data space for data move instructions (MVPD and READA). This capability, in conjunction with the feature of dual-operand read, supports the execution of single-cycle, 3-operand instructions such as the FIRS instruction.

The C54x DSP also has an on-chip bidirectional bus for accessing on-chip peripherals. This bus is connected to DB and EB through the bus exchanger in the CPU interface. Accesses that use this bus can require two or more cycles for reads and writes, depending on the peripheral's structure.

Table 2–1 summarizes the buses used by various types of accesses.

Q2. Donner la signification et le rôle de chacun des bus représentés dans le schéma.

L'étude de ce pipeline est en rapport directe avec l'organisation logicielle que le programmeur va implanter dans un processeur de ce type. En effet, la multiplicité des bus donne des accès particuliers aux différentes mémoires qu'il conviendra d'optimiser pour une application.

Le paragraphe ci-dessous provient aussi de la documentation pour introduire l'explication du pipeline de ce processeur.

2.6 Pipeline Operation

An instruction pipeline consists of a sequence of operations that occur during the execution of an instruction. The C54x™ DSP pipeline has six levels: *prefetch*, *fetch*, *decode*, *access*, *read*, and *execute*. At each of the levels, an independent operation occurs. Because these operations are independent, from one to six instructions can be active in any given cycle, each instruction at a different stage of completion. Typically, the pipeline is full with a sequential set of instructions, each at one of the six stages. When a PC discontinuity occurs, such as during a branch, call, or return, one or more stages of the pipeline may be temporarily unused. For more details about the pipeline operation, see Chapter 7, *Pipeline*.

Q3. Quelle est la profondeur de ce pipeline ?

Q4. Expliquer le type d'aléa qui est cité comme exemple dans ce paragraphe.

Q5. Dans l'explication ci-dessous, expliquer ce qu'est une situation de conflit ? Quelle solution logicielle Texas Instrument donne pour résoudre ce conflit simplement.

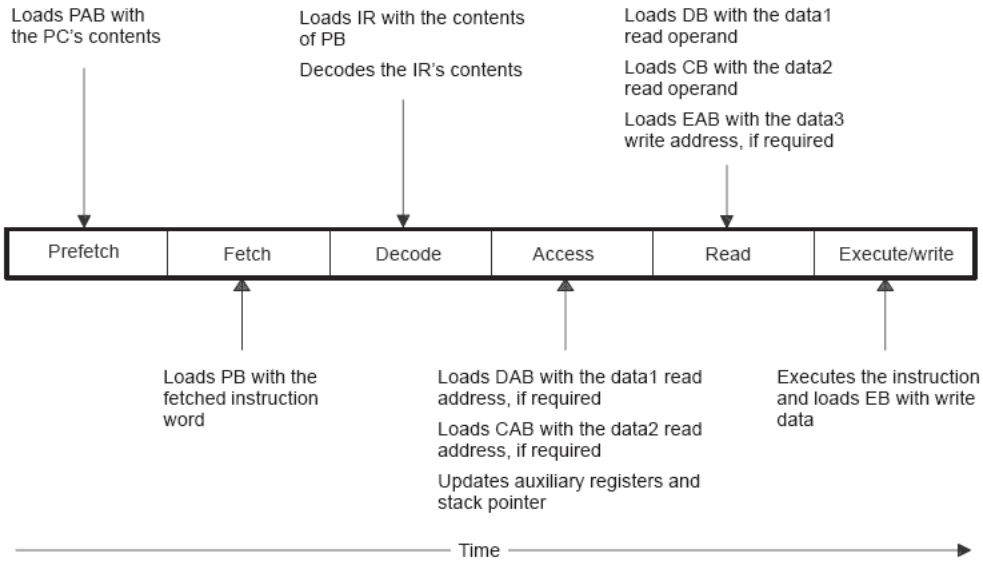
7.5 Pipeline Latencies

The C54x™ DSP pipeline allows multiple instructions to access CPU resources simultaneously. Because CPU resources are limited, conflicts can occur when one CPU resource is accessed by more than one pipeline stage. Some of these pipeline conflicts are resolved automatically by the CPU by delaying accesses. Other conflicts are unprotected and must be resolved by the programmer.

In general, unprotected conflicts are resolved by rearranging instructions or by inserting NOP instruction (no operation performed). They can also be avoided by using only instructions that do not create any pipeline conflicts or by observing necessary delays before certain registers are accessed.

Ci-dessous sont représentés les différents étages.

Figure 7–1. Pipeline Stages



Note : IR = Instruction Register

Q6. Expliquer en quelques mots chacun des rôles des étages de ce pipeline.

Q7. Les bus DAB, et CAB sont en fait reliés à la même mémoire appelé DARAM. En lisant la partie de la documentation ci-dessous, expliquer comment il est possible d'utiliser la même ressource mémoire sans créer d'attente dans le pipeline.

> It is assumed that all memory accesses in the figure are performed by single-cycle, single-word instructions to on-chip dual-access memory. The on-chip dual-access memory can actually support two accesses in a single pipeline cycle. This is discussed in section 7.3, *Dual-Access Memory and the Pipeline*, on page 7-27.

Université de Savoie
Module I621
Architecture des Ordinateurs

TD 3
Mémoires Caches

I. Rappels de cours

1. *Le principe de localité.*

Le principe de localité établit que les programmes accèdent à un instant donné à une part limitée de l'ensemble de l'espace adressable. Par analogie, lorsqu'on travaille dans une bibliothèque, nous avons besoin d'accéder à une quantité très limitée de livre à un instant donné.
Il y a deux types de localité, l'un est appelées « localité temporelle » et l'autre « localité spatiale ».

1.1. Localité temporelle (dans le temps) :

Si un élément est référencé, il aura tendance à être référencé de nouveau (Analogie : le livre que j'ai ramené il y a peu de temps va sûrement me resservir). C'est le cas des boucles de programme car elles appellent plusieurs fois les mêmes instructions dans un intervalle de temps réduit.

1.2. Localité spatiale (dans l'espace) :

Si un élément est référencé, les éléments dont les adresses sont voisines auront tendance à être référencés bientôt. (Analogie : le livre juste à coté de celui que j'ai utilisé à plus de chance de m'intéresser que celui d'une autre étagère). L'exemple le plus courant est l'accès à un tableau. En effet, il y a de forte chance d'accéder à des cases consécutives d'un même tableau.

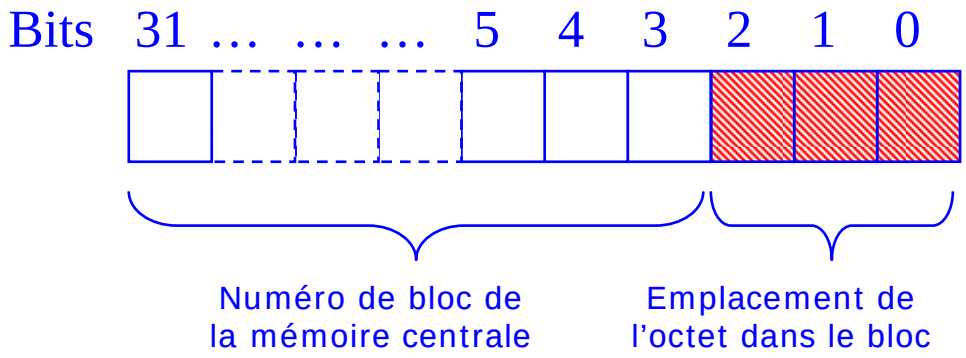
2. *La hiérarchie mémoire*

La construction de la mémoire sous forme d'une hiérarchie à plusieurs niveaux s'avère avantageuse du fait des différences de temps d'accès. La mémoire la plus rapide (donc la plus chère) est la plus proche du processeur, la mémoire la plus lente (donc la moins chère) est placée à un niveau inférieur.
Les performances de cette hiérarchie de mémoires sont mesurées à l'aide du taux de succès (si la valeur demandée par le processeur est présente) et du taux de défauts. Le coût d'un défaut est le temps nécessaire pour remplacer un bloc (élément d'information) du niveau supérieur au niveau inférieur.

3. *Rappels sur la mémoire cache*

Le cache est le nom qui a été choisi pour représenter le niveau de la hiérarchie mémoire situé entre le processeur et la mémoire principale dans le but de tirer parti de la localité des accès. L'unité de transfert entre la mémoire centrale et la mémoire cache est le bloc. Un bloc est un ensemble d'octets.

Une mémoire cache est structurée en lignes. Chaque ligne contient un bloc. Chaque bloc contient des mots dont les adresses sont consécutives en mémoire centrale. La partie haute de l'adresse est donc celle correspondant au numéro de bloc, et la partie basse, celle correspondant à l'emplacement du mot dans la ligne de cache.



Une ligne du cache est soit totalement inoccupée soit totalement pleine.

II. Cache à correspondance directe

1. Evolution du remplissage d'une mémoire cache

Un processeur 8 bits possède un cache à correspondance directe de 8 lignes de 4 octets.

Q1. Quelle est la taille des blocs ?

La série de requêtes (adresse sur 8 bits) envoyé par le processeur est la suivante (en hexadécimal) : 0x22, 0x23, 0x4, 0x16, 0x5, 0x4, 0x26, 0x18. Les tableau 2 représentent une mémoire cache à correspondance directe.

Q2. Donner la définition de tous les champs (colonnes du Tableau 1) de la mémoire cache.

Q3. Préciser le découpage de l'adresse binaire 8 bits pour le fonctionnement de ce cache. (Etiquette, numéro de bloc,…).

Le tableau ci-dessous donne le début du contenu de la mémoire centrale pour chaque adresse.

@	Donnée
00h	05h
01h	AEh
02h	BCh
03h	FFh
04h	23h
05h	27h
06h	BDh
07h	99h
08h	69h
09h	C7h

@	Donnée
0Ah	EEh
0Bh	40h
0Ch	74h
0Dh	75h
0Eh	77h
0Fh	05h
10h	AEh
11h	37h
12h	01h
13h	79h

@	Donnée
14h	10h
15h	03h
16h	AEh
17h	BCh
18h	FFh
19h	23h
1Ah	37h
1Bh	98h
1Ch	10h
1Dh	03h

@	Donnée
1Eh	05h
1Fh	AEh
20h	FEh
21h	FEh
22h	10h
23h	03h
24h	05h
25h	AEh
26h	28h
27h	36h

Q4. Pour chaque requête, complétez le contenu du Tableau 1 et du Tableau 2.

@ demandée (hexa)	Adresse binaire	Succès / défaut	Numéro de bloc du cache assigné	Numéro de bloc de la mémoire centrale
22h				
23h				
04h				
16h				
05h				
04h				
26h				
18h				

Tableau 1 : Succès/ Défaut de chaque requête et blocs assignés

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Etiquette	V	Donnée	

Tableau 2 : Evolution du remplissage de la mémoire cache après chaque requête

Q5. Commenter l'efficacité du cache à correspondance directe.

III. Schéma d'une mémoire cache

On représente une mémoire cache sous forme de tableau (comme dans l'exercice précédent).

Q1. Donner le schéma du cache à correspondance directe correspondant au format de l'adresse suivante.

Etiquette	Index	Adresse octet
31 8	7 4	3 0

IV. Comparaison des types de cache

On dispose de deux mémoires caches, chacune d'elles possédant quatre blocs. Une des mémoires caches est totalement associative et la seconde est à correspondance directe.
La technique de remplacement des blocs est celle du bloc LRU (Least Recently Used).

Q1. Parmi les deux types de mémoires caches, laquelle est concernée par cette technique de remplacement de blocs ?

La séquence de numéro de blocs demandé par le processeur est la suivante : 0, 8, 0, 6,8, 5, 7 8, 0.

Q2. Déterminer le nombre de défauts pour chaque organisation de cache.

Q3. Commenter l'efficacité de l'associativité.

Q4. Vu son efficacité, pourquoi n'utilise t on pas toujours ce type de mémoire ?

V. Taille totale d'une mémoire cache

Un processeur possède 32 bits d'adresse et des blocs de 4 octets. Sa mémoire cache a correspondance directe est de 64 Ko de données.

Q1. Quel est le nombre de bits pour l'index et pour l'étiquette ?

Q2. Quel est le nombre total de bits requis pour ce cache? Faite un rapport entre le nombre de bit utile (données) et le nombre de bit total.

VI. Correspondance des adresses

Considérons un cache à correspondance directe de 64 blocs et une taille de bloc de 16 octets.

Q1. A quel numéro de bloc en mémoire cache, l'adresse processeur 1200 correspond elle?

Q2. A quel numéro de ligne du cache (index) cela correspond t il ?

VII. Rôle du programmeur

1. Définition donnée par la documentation

Voici une définition tirée de l'aide de Code Composer Studio (Outils de Développement pour processeur Texas Instrument).

Cache block :
A section of cache memory. Each block has an associated tag register and is divided into four subblocks. Cache memory is allocated in block-size portions, but cache servicing is performed at the subblock level, with subblocks brought in as needed.

Q1. Expliquer précisément les termes employés dans la définition du « cache block »

2. 1er exemple d'une situation de conflit

The following example illustrates how capacity misses in instruction cache (I-Cache) can be eliminated by breaking the algorithm into smaller pieces that fit in I-Cache.
Consider the code example below:

Example:

```
1. for (i=0; i<N; i++){
2. function_1();
3. function_2();
4. }
```

Assume that the combined code size of function_1 and function_2, as shown in example above, ***is larger than the size of I-Cache.*** *Since the capacity of the cache is not sufficient to hold all functions in the loop, the loop may have to be split up in order to achieve code reuse without evictions.*

Q2. Donner la solution que vous apporteriez d'après les conseils de l'aide. Réécrivez le code en implémentant votre solution.

Q3. Quelle mise en garde est formulée dans la Remarque ci dessous :

However, the temporary buffer tmp[] now has to hold all intermediate results from each call to function_1.

```
1. for (i=0; i<N; i++){
2. function_1(in[i], tmp[i]);
3. }
```

```
4.
5.  for (i=0; i<N; i++){
6.  function_2(tmp[i], out[i]);
7.  }
```

---

3. 2ème cas de conflit

On considère maintenant un nouveau cas. Cette fois la totalité des deux fonctions entres dans l'espace mémoire cache. Un autre problème exposé ci-dessous peut se poser. Nous travaillons avec une mémoire cache **à correspondance directe**.

The following example illustrates how the conflict misses in the instruction cache (I-Cache) can be eliminated by allocating the code contiguously in memory.
Consider the I-Cache Conflicts Code example below:
Example:

```
1.  for (i=0; i<N; i++){
2.  function_1();
3.  function_2();
4.  function_3();
5.  }
```

Assume that function_1, function_2 and function_3 have been placed by the linker such that they overlap in I-Cache. When they are called consecutively in a loop, they evict each other from cache before the functions are called again. For every iteration, each function call causes cache misses. These misses are known as conflict misses. They can be completely avoided by allocating the code of the two functions into non-conflicting sets. The most straightforward way this can be achieved is to place the code of the two functions contiguously in memory.

To avoid this, the function that require contiguous placement may be assigned individual sections by using the pragma CODE_SECTION before the definition of the functions:

```
1.  #pragma CODE_SECTION(function_1,".funct1")
2.  #pragma CODE_SECTION(function_2,".funct2")
3.  #pragma CODE_SECTION(function_3,".funct3")
4.
5.  void function_1(){...}
6.  void function_2(){...}
7.  void function_3(){...}
```

The linker command file would then be specified as:

```
SECTIONS{
.vectors > vecs          .cinit > SRAM
.funct1 > SRAM
.funct2 > SRAM
```

.funct3 > SRAM
.text > SRAM
.stack > SRAM
...}

The functions that should be considered for reordering are those that are repeatedly called within the same loop, or within some time frame.

Q4. Analyser le problème et préciser de quelle façon il est résolu ici.

Université de Savoie
Module I621 : Architecture des Ordinateurs

TD 4

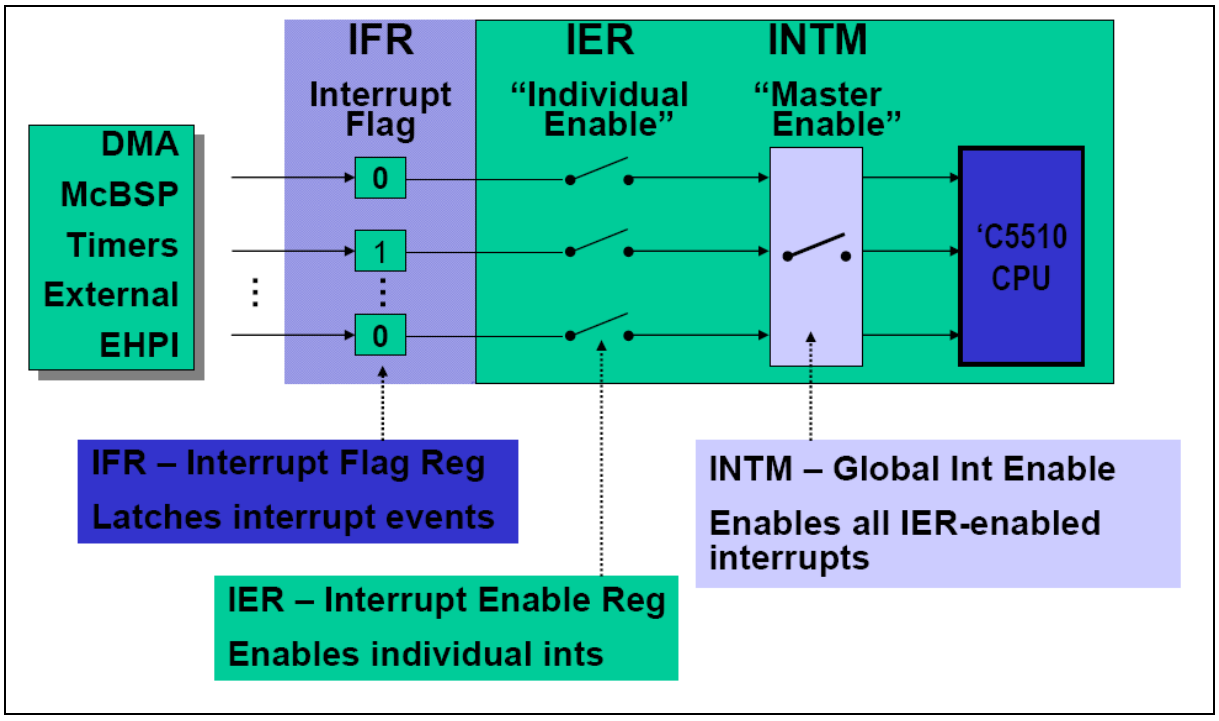
Les interruptions

I.Interruption dans un DSP TMS320

Le schéma récapitulatif du fonctionnement des interruptions dans un DSP TMS320 est donné ci-dessous.

Notes :

- DMA : Direct Memory Access
- McBSP : Multi-channel Buffered Serial Port



1. Masque d'interruption

- Maskable interrupts. These are hardware or software interrupts that can be blocked (masked) or enabled (unmasked) using software. The C54x DSP supports up to 16 user-maskable interrupts (SINT15–SINT0). Each device uses a subset of these 16 interrupts. For example, the C541 uses only nine of these interrupts (the others are tied high internally). Some of these have two names because they can be initiated by software or hardware; for the C541, the hardware names for these interrupts are:
  - $\overline{\text{INT3}}$ through $\overline{\text{INT0}}$
  - RINT0, XINT0, RINT1, and XINT1 (serial port interrupts)
  - TINT (timer interrupt)
- Nonmaskable interrupts. These interrupts cannot be blocked. The C54x DSP always acknowledges this type of interrupt and branches from the main program to an ISR. The C54x DSP nonmaskable interrupts include all software interrupts and two external hardware interrupts: $\overline{\text{RS}}$ (reset) and $\overline{\text{NMI}}$. ($\overline{\text{RS}}$ and $\overline{\text{NMI}}$ can also be asserted using software.)

6.10.2 Interrupt Mask Register (IMR)

Figure 6–3 shows how the C54x DSP uses a memory-mapped IMR for masking external and internal interrupts. If INTM = 0 in ST1, a 1 in any IMR bit enables the corresponding interrupt. Neither $\overline{\text{NMI}}$ nor $\overline{\text{RS}}$ is included in the IMR, because IMR has no effect on these interrupts. You can read or write to the IMR.

Figure 6–3. Interrupt Mask Register (IMR) Diagram

(a) C541 IMR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	XINT1	RINT1	XINT0	RINT0	TINT	INT2	INT1	INT0

Q1. Expliquer ce qu'est un masque d'interruption et le rôle du registre IMR.

2. Flag d'interruption

6.10.1 Interrupt Flag Register (IFR)

IFR is a memory-mapped CPU register that identifies and clears active interrupts (see Figure 6–2). An interrupt sets its corresponding interrupt flag in IFR until it is recognized by the CPU.

Figure 6–2. Interrupt Flag Register (IFR) Diagram

(a) C541 IFR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	XINT1	RINT1	XINT0	RINT0	TINT	INT2	INT1	INT0

Q2. Expliquer ce qu'est un flag d'interruption et le rôle du registre IFR.

3. Vecteur d'interruption

6.10.10 Interrupt Tables

Table 6–19 through Table 6–24 show the interrupt trap number, priority, and location for some C54x devices.

Table 6–19. TMS320C541 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location (Hex)	Function
0	1	$\overline{RS}$/SINTR	0	Reset (hardware and software reset)
1	2	$\overline{NMI}$/SINT16	4	Nonmaskable interrupt
2	–	SINT17	8	Software interrupt #17
3	–	SINT18	C	Software interrupt #18
4	–	SINT19	10	Software interrupt #19
5	–	SINT20	14	Software interrupt #20
6	–	SINT21	18	Software interrupt #21
7	–	SINT22	1C	Software interrupt #22
8	–	SINT23	20	Software interrupt #23
9	–	SINT24	24	Software interrupt #24
10	–	SINT25	28	Software interrupt #25
11	–	SINT26	2C	Software interrupt #26
12	–	SINT27	30	Software interrupt #27
13	–	SINT28	34	Software interrupt #28
14	–	SINT29	38	Software interrupt #29; reserved
15	–	SINT30	3C	Software interrupt #30; reserved
16	3	$\overline{INT0}$/SINT0	40	External user interrupt #0
17	4	$\overline{INT1}$/SINT1	44	External user interrupt #1
18	5	$\overline{INT2}$/SINT2	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	RINT0/SINT4	50	Serial port 0 receive interrupt
21	8	XINT0/SINT5	54	Serial port 0 transmit interrupt
22	9	RINT1/SINT6	58	Serial port 1 receive interrupt
23	10	XINT1/SINT7	5C	Serial port 1 transmit interrupt
24	11	$\overline{INT3}$/SINT8	60	External user interrupt #3
25–31	–		64–7F	Reserved

Q3. Que va- t-on trouver aux adresse spécifiées par la colonne « location » ?

4. Programmation des interruptions

Phase 3: Execute Interrupt Service Routine (ISR)

After acknowledging the interrupt, the CPU:

1) Stores the program counter (PC) value (the return address) to the top of the stack in data memory
2) Loads the PC with the address of the interrupt vector
3) Fetches the instruction located at the vector address. (If the branch is delayed and you also stored one 2-word instruction or two 1-word instructions, the CPU also fetches these words.)
4) Executes the branch, which leads it to the address of your ISR. (If the branch is delayed, the additional instruction(s) are executed before the branch.)
5) Executes the ISR until a return instruction concludes the ISR
6) Follows the stack pointer (SP) to the top of the stack, and pops the return address off the stack and into PC
7) Continues executing the main program

En programmation C, vous n'avez pas à gérer l'ensemble des ces points. Par exemple la sauvegarde du contexte, et le chargement du PC est fait automatiquement.

Q4. Donner les configurations en C que vous feriez pour programmer une interruption du timer 0.

Université de Savoie
Module I621 : Architecture des Ordinateurs

TD 4

Les transferts DMA

I. DMA dans un DSP

3.1 DMA Overview

The direct memory access (DMA) controller transfers data between regions in the memory map without intervention by the CPU. The DMA allows movement to and from internal memory, internal peripherals, or external devices to occur in the background of CPU operation. The DMA has six independent programmable channels allowing six different contexts for DMA operation. The DMA controller also services requests from the host port interface peripherals (HPI-8 or HPI-16) to utilize the DMA bus. Throughout this chapter, the abbreviation HPIx is used to denote either the HPI-8 or HPI-16.

The terms used in discussing DMA operations are defined as follows:

- **Read transfer**. The DMA reads a data element from a source location in memory. The source may be memory or a peripheral device, and may be located in program, data, or I/O space.
- **Write transfer**. The DMA writes the data elements that were read during a read transfer to its destination in memory location. The destination may be memory or a peripheral device, and may be located in program, data, or I/O space.
- **Element transfer**. The combined read and write transfer for a single data element.
- **Frame transfer**. Each DMA channel has an independently programmable number of elements per frame. In completing a frame transfer, the DMA moves all elements in a single frame.
- **Block Transfer**. Each DMA channel also has an independently programmable number of frames per block. In completing a block transfer, the DMA moves the total number of frames defined for the block.

Q1. Combien de canaux DMA possèdent ce contrôleur ? Est-ce plus contraignant pour un processeur (du point de vue de l'utilisation de ces ressources) de posséder plus de canaux.

Q2. En étudiant les registres du processeur ci-dessous, donner les actions simples que doit fournir le processeur afin de faire fonctionner chacun des canaux DMA. (seuls les canaux 0 et

1 sont affichés). Préciser seulement ce qui vous semble important pour la compréhension générale des transferts DMA.

—	00h	DMSRC0	Channel 0 Source Address Register
—	01h	DMDST0	Channel 0 Destination Address Register
—	02h	DMCTR0	Channel 0 Element Count Register
—	03h	DMSFC0	Channel 0 Sync Select and Frame Count Register
—	04h	DMMCR0	Channel 0 Transfer Mode Control Register
—	05h	DMSRC1	Channel 1 Source Address Register
—	06h	DMDST1	Channel 1 Destination Address Register
—	07h	DMCTR1	Channel 1 Element Count Register
—	08h	DMSFC1	Channel 1 Sync Select and Frame Count Register
—	09h	DMMCR1	Channel 1 Transfer Mode Control Register

3.2.3.1 Source and Destination Address Registers

Each DMA element transfer involves a read followed by a write. The address of the data to be read is called the source address, and the address where the data will be written is called the destination address.

3.2.3.2 Element Count Register

Each DMA Channel has a 16-bit element counter that keeps track of the number of DMA transfers to be performed. The element counter is initialized with the 16-bit unsigned number contained in the DMA channel element count register (DMCTRn) that represents the number of elements to be transferred. The element count register should be initialized with one less than the desired number of element transfers. For example, if DMA channel 2 is to move nine data-elements, DMCTR2 should be initialized to eight.

Le transfert DMA peut être associé aux mécanismes d'interruption du processeur.

Q3. Rappeler quel événements peut interrompre le processeur afin de rendre très performant le transfert DMA ?

Dans les processeurs possédant un contrôleur DMA, il est possible de choisir précisément l'instant de l'interruption du processeur lors d'un transfert.

Q4. Préciser les 4 interruptions possibles, et expliquer d'un point de vue de l'algorithmique pourquoi il peut être intéressant de travailler dans certains de ces modes ?

13	IMOD	0	DMA interrupt generation mode bit. In ABU mode (CTMOD = 1): IMOD = 0 Interrupt at buffer full only IMOD = 1 Interrupt at half buffer full and buffer full In multiframe mode (CTMOD = 0): IMOD = 0 Interrupt at completion of block transfer IMOD = 1 Interrupt at end of frame and end of block