Gameboy hardware

Plan d’adressage de la Gameboy

Mémoire générale	
00000000-00003FFFF	BIOS – système ROM (16 KBytes)
00004000 – 01FFFFFFF	Pas utilisé
02000000 – 0203FFFF	WRAM – On Board (256 KBytes, Work RAM)
02040000 – 02FFFFFF	Pas utilisé
03000000 – 03007FFF	WRAM – In-Chip (32 KBytes)
03008000 – 03FFFFFF	Pas utilisé
04000000 – 040003FE	I/O Registers
04000400 – 04FFFFFF	Pas utilisé

Plan d’adressage de la Gameboy

Mémoire externe Affichage	
05000000-050003FF	BG/OBJ Palette RAM (1 KByte)
05000400 – 05FFFFFF	Pas utilisé
06000000 – 06017FFF	VRAM – Video RAM (96 KBytes)
06018000 – 06FFFFFF	Pas utilisé
07000000 – 0700003FF	OAM – OBK Attributes (1 KByte)
07000400 – 07FFFFFF	Pas utilisé

10000000 – FFFFFFFF: pas utilisé, 4 bits de poids fort non décodé

Bus, temps d’attente

Region	Bus	Lecture	Ecriture	Cycles
BIOS ROM	32	8/16/32	-	1/1/1
WRam 32K	32	8/16/32	8/16/32	1/1/1
I/O	32	8/16/32	8/16/32	1/1/1
WRam 256K	16	8/16/32	8/16/32	3/3/6
VRAM	16	8/16/32	8/16/32	1/1/2

Remarques

La table des interruptions du processeur ARM ne contient pas l'adresse de la routine à exécuter. Elle contient une instruction que le processeur exécute (généralement un branchement B).

Lorsqu'une interruption est levée:

- le processeur suspend l'exécution en cours (fini d'exécuter l'instruction courante)
- charge le pc à l'adresse correspondant à l'interruption levée

Les adresses correspondantes correspondent à la table des interruptions (Reset 0x0, Undefined instruction 0x4, SWI 0x8, Prefetch abort 0xC, Data abort 0x10, IRQ 0x18, FIQ 0x1C)

Remarques

Pour traiter une interruption particulière il faut

- Ecrire le gestionnaire d'interruption
- dans la table d'interruption mettre à la bonne adresse une instruction B *adresse_gestionnaire, LDR pc, [pc, #offset].*

La table des interruptions débute à l'adresse 0x00000000, qui se trouve en mémoire ROM et n'est donc pas accessible!

Le programme en BIOS (BIOS interrupt handler) va chercher l'adresse du gestionnaire utilisateur à l'adresse **0x3007FFC** (quelle que soit l'interruption levée).

Pour intercepter une interruption il faut donc placer l'adresse du gestionnaire d'interruption à cette adresse.

Link Register lr

Lorsqu'une interruption est levée le registre lr est modifié. Son contenu dépend du type d'interruption levée:

Reset	---	
Data abort	lr - 8	l'instruction qui a levé l'exception
FIQ	lr - 4	adresse de retour du gestionnaire
IRQ	lr – 4	adresse de retour du gestionnaire
Prefetch abort	lr – 4	l'instruction qui a levé l'exception
SWI	lr	la prochaine instruction après SWI
Undefined instruction	lr	la prochaine instruction après l'instruction non définie

Exemple

```
handler
        <code>
        subs    pc, r14, #4         pc = r14 -4 + restauration de cpsr

handler
        sub     r14, r14, #4
        <code>
        movs    pc, r14             return spsr -> cpsr

handler
        sub     r14, r14, #4
        stmfd   r13!, {r0-r3, r14}
        <code>
        ldmfd   r13!, {r0-r3,pc}^   return + spsr -> cpsr
```

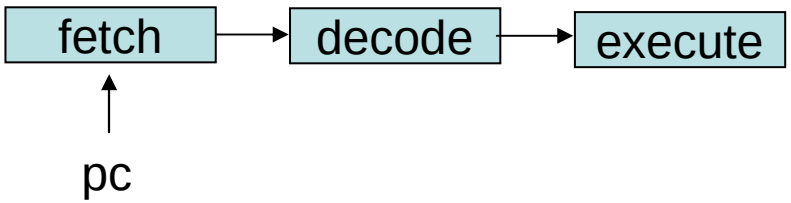
Le gestionnaire d'interruption du BIOS - GBA

IRQ

```
00000018    b       0x128           instruction branchement gestionnaire

00000128    stmfd   r13!, {r0-r3,r12,r14}
0000012C    mov     r0, 0x4000000   = 3FFFFFC +4
00000130    add     r14, r15,0x0    adresse de retour 0x138
00000134    ldr     r15,  [r0, -4]  saute à l'adresse [03FFFFFC] = [03007FFC]
00000138    ldmfd   r13!, {r0-r3,r12,r14}
0000013C    subs    r15, r14, #4
```

à cause du s et du fait que le pc est la destination
cpsr <- spsr



Priorités

Les interruptions possèdent des niveaux de priorités qui déterminent l'ordre d'exécution lorsque plusieurs interruptions sont levées simultanément

Reset -> Data abort -> fiq -> irq -> Prefetch abort -> SWI -> Undefined instruction

Les interruptions de la GBA

Registre REG_IE (0x4000200)	
Bit	**Description**
0	VB – Vertical blank interrupt
1	HB – Horizontal blank interrupt
2	VC – vertical scanline count interrupt
3	T0 – timer 0
4	T1 – timer 1
5	T2 – timer 3
6	T3 – timer 4
7	COM – serial communication interrupt
8	DMA0 – DMA0 finished interrupt
9	DMA1
10	DMA2
11	DMA3
12	BUTTON – button interrupt
13	CART – game cartridge interrupt

Les interruptions de la GBA

Un programme C contiendra les définitions suivantes:

```
#define INT_VBLANK 0x0001,
#define INT_HBLANK 0x0002,
#define INT_VCOUNT 0x0004,
#define INT_TIMER0 0x0008,
…
#define INT_TIMER3 0x0040,
#define INT_COM 0x0080
#define INT_DMA0 0x0100
…
#define INT_DMA3 0x0800
#define INT_BUTTON 0x1000
#define INT_CART 0x2000
```

les interruptions de la GBA

Le registre **REG_IME** (Interrupt Master Enable, 0x4000208) permet d'activer toutes les interruptions (=0x1) ou inhiber les interruptions (=0x0).

Le registre **REG_IE** permet d'activer/inhiber les interruptions particulières.

Le registre **REG_IF** (0x4000202) permet de déterminer quelle interruption à été levée, le bit correspondant étant positionné à 1. la signification des différents bits est la même que celle du registre REG_IE (un et un seul bit et positionné à la fois). On acquitte le traitement de l'interruption en écrivant un 1 dans ce registre à la position correspondant à l'interruption (d'où le mot clé *volatile).*

```
if ((REG_IF & INT_TIMER0) == INT_TIMER0) {
        <gestionnaire de l'interruption TIMER0>
}
```

Les interruptions de la GBA

Lorsqu'on désire inhiber une interruption, il faut d'abord inhiber le bit correspondant du registre **REG_IME** et ensuite le bit correspondant du registre **REG_IE**.

Registre d’état vidéo

Registre REG_DISPSTAT (0x4000004)	
Bit	**Description**
0	VB – vertical blank occuring (read only)
1	HB – horizontal blank occuring (read only)
2	V Counter flag (read only)
3	vblank interrupt enable
4	hblank interrupt enable
5	vcount interrupt enable
6 - 15	vcount – vertical count value

Exemple

On veut écrire un programme qui lève une interruption chaque fois qu'un rafraichissement horizontal de l'écran est terminé.

```
int main(void)
{
    SetMode(3 | BG2_ENABLE);                    // Mode 3
    REG_IME = 0x0;                              // disable interrupt
    REG_INTERRUPT = (u32)MyHandler;
    REG_IE |= INT_HBLANK;                       // enable HBLANK interrupt, bit 4
    REG_DISPSTAT |=0x10;                        // HBLANK Status
    REG_IME = 0x1;                              // enable interrupt

    while(1);
    return 0;
}
```

principaux registres pour les interruptions

```
// interrupt registers
#define REG_IME *(u16*)0x4000208

#define REG_IE  *(u16*)0x4000200

#define REG_IF  *(volatile u16*)0x4000202

#define REG_INTERRUPT   *(u32*)0x3007FFC

#define REG_DISPSTAT    *(u16*)0x4000004
```

Exemple

```
void MyHandler(void)
{
    u16 Int_Flag;  u16 x, y;  u16 color;

    REG_IME = 0x0;              // disable interrupt
    Int_Flag = REG_IF;          // backup flag register

    // look for horizontal refresh
    if((REG_IF | INT_HBLANK) == INT_HBLANK)
    {
        x = rand() % 240; y = rand() % 160;
        color = RGB(rand()%31, rand()%31, rand()%31);
        drawPixel3(x, y, color);
    }
    REG_IF = Int_Flag;          // restore flags
    REG_IME = 0x1;              // restore interrupt
}
```

Boutons et interruptions

Les boutons sont accessibles via deux registres:
0x4000132 KEYCNT = Key Interrupt Controller (lecture/écriture)
permet d'activer/inhiber les interruptions pour le clavier

Registre KEYCNT (0X4000132)			
Bit	Description	Bit	Description
0	A (0=ignore, 1= sélectionne)	7	Down
1	B	8	R
2	Select	9	L
3	Start	10 – 13	inutilisé
4	Right	14	IRQ (0=disable, 1=enable)
5	Left	15	IRQ condition (0=au moins un bouton, 1=tout les boutons
6	Up		

Boutons et interruptions

le deuxième registre est

0x4000130 KEYINPUT permet de lire si le bouton correspondant est pressé (=0) ou relâché (=1).

Exemple

```
#define KEYCNT  *(volatile u16*)0x4000132
int main(void)
{
   REG_IME = 0x0;
   REG_INTERRUPT = (u32)MyHandler;   //0x3007FFFF
   SetMode(3 | BG2_ENABLE);
   REG_IE |= INT_BUTTON;          // autorise les interruptions boutons



    KEYCNT |= 0x7FFF;
    REG_IME = 0x1;                 // enable interrupt

  while(1)
  { }
   return 0;
}
```

Exemple

```
void MyHandler(void)
{
    REG_IME = 0x0;     // desactive les interruptions

   DrawBox3(140, 30, 180, 80, 0x0);    // efface l'écran

    sprintf(str, "%i", cpt++);                  // formate la chaine
    Print(140,30,str,0x0FF0);
    REG_IF=REG_IF;                              // acquitte le traitement de l'interruption
    REG_IME=0x1;
}
```

Les timers

Un timer consiste principalement en 1 registre de 16 bits qui s'incrémente à une fréquence fixe. La GBA dispose de 4 timers dont la fréquence peut être choisie parmi:

1. 16.78 MHz, chaque 59.95 nanosecondes
2. 64 cycles, 3.8 microsecondes
3. 256 cycles, 15.6 microsecondes
4. 1024 cycles, 61 microsecondes

Pour contrôler le fonctionnement du timer on utilise les registres (32bits) REG_TM0CNT, REG_TM1CNT,…, REG_TM3CNT situés aux adresses 0x4000100, 0x4000104, 0x4000108 et 0x400010C

0x4000100	REG_TM0CNT_L	REG_TM0CNT_H
0x4000104	REG_TM1CNT_L	REG_TM1CNT_H
0x4000108	REG_TM2CNT_L	REG_TM2CNT_H
0x400010C	REG_TM3CNT_L	REG_TM3CNT_H

Contrôle des timers

REG_TMxCNT_H	
Bits	**Description**
0-1	fréquence (f, f/64, f/256, f/1024)
2	incrément lorsque le précédent compteur overflow
3-5	inutilisés
6	génère une interruption lorsqu'il y a overflow
7	active (enable) timer
8-15	inutilisés

Contrôle des timers

Les registres REG_TMxCNT_L (aussi REG_TMxD) contiennent la valeur d'initialisation des registres. Apres que le registre 'compte' 65535' (overflow), il s'initialise avec la valeur contenue dans REG_TMxCNT (pas nécessairement 0).

Le timer est initialisé lorsque le bit d'activation (bit 7 REG_TMxCNT_H) est activé (passe de 0 à 1).

On peut activer le timer et initialiser le registre REG_TMxCNT_L simultanément.

Exemple:

```
REG_TM0D = 30000;
REG_TM0CNT |= TIMER_FREQUENCY_256 | TIMER_ENABLE;
```

Le compteur compte: 30000, 30001, …., 65535, 30000, 30001, …

Résumé

REG_IE (0x4000200) permet de définir les événements susceptibles de lever une interruption

REG_IME (0x4000208) permet de masquer/autoriser toutes les interruptions

des registres spécifiques à chaque interruption:

- Affichage: DISPSTAT (0x4000004) pour VB, HB, VC
- Timer: REG_TMxCNT (0c4000100, 0x4000104, 0x4000108, 0x400010C)
- Boutons: KEYCNT (0x40000134)

Résumé

le gestionnaire d'interruption se trouve à l'adresse contenue en 0x3007FFC (0x3007FFC est un pointeur sur une fonction)

Dans le gestionnaire, il faut

1. Inhiber les interruptions pour ne pas être interrompu
2. Exécuter le code correspondant au traitement de l'interruption. Le même gestionnaire est activé pour toutes les interruptions, il doit donc déterminer l'interruption levée et effectuer le traitement adéquat si nécessaire. Le registre REG_IF (0x4000202) qui a la même structure que le registre REG_IE permet de déterminer l'événement.
3. Effectuer REG_IF = REG_IF pour désactiver l'interruption
4. Activer les interruptions

Les fonctions du BIOS

Le BIOS contient des fonctions qu'il est possible d'appeler en utilisant l'instruction

SWI numéro_fonction

A l'appel d'une fonction BIOS, spsr, r11, r12, r14 sont sauvé sur la pile superviseur (_svc) et le gestionnaire d'interruption SWI passe en mode Système et utilise la pile utilisateur.

GBA	Function
00h	SoftReset
01h	RegisterRamReset
02h	Halt
03h	Stop/Sleep
04h	IntrWait
05h	VBlankIntrWait
06h	Div
07h	DivArm
08h	Sqrt
09h	ArcTan
0Ah	ArcTan2
0Bh	CpuSet
0Ch	CpuFastSet
0Dh	GetBiosChecksum
0Eh	BgAffineSet
0Fh	ObjAffineSet
10h	BitUnPack
11h	LZ77UnCompWram
12h	LZ77UnCompVram
13h	HuffUnComp
14h	RLUnCompWram
15h	RLUnCompVram
16h	Diff8bitUnFilterWram
17h	Diff8bitUnFilterVram
18h	Diff16bitUnFilter
19h	SoundBias
1Ah	SoundDriverInit
1Bh	SoundDriverMode
1Ch	SoundDriverMain
1Dh	SoundDriverVSync
1Eh	SoundChannelClear
1Fh	MidiKey2Freq
20h	SoundWhatever0
21h	SoundWhatever1
22h	SoundWhatever2
23h	SoundWhatever3
24h	SoundWhatever4
25h	MultiBoot
26h	HardReset
27h	CustomHalt
28h	SoundDriverVSyncOff
29h	SoundDriverVSyncOn
2Ah	SoundGetJumpList

Fonction arithmétiques

Div
DivArm
Sqrt
ArcTan
ArcTan2

Div, SWI 0x06 division signée r0/r1
entrée:
    r0: numérateur
    r1: dénumerateur
sortie:
    r0: division entière signée
    r1: reste signé
    r3: valeur absolue de la division

Fonctions arithmétiques

Sqrt, SWI 0x7 sqrt(r0)

entrée:

r0: nombre non signé entier sur 32 bits

sortie

r0: nombre non signé entier sur 16 bits

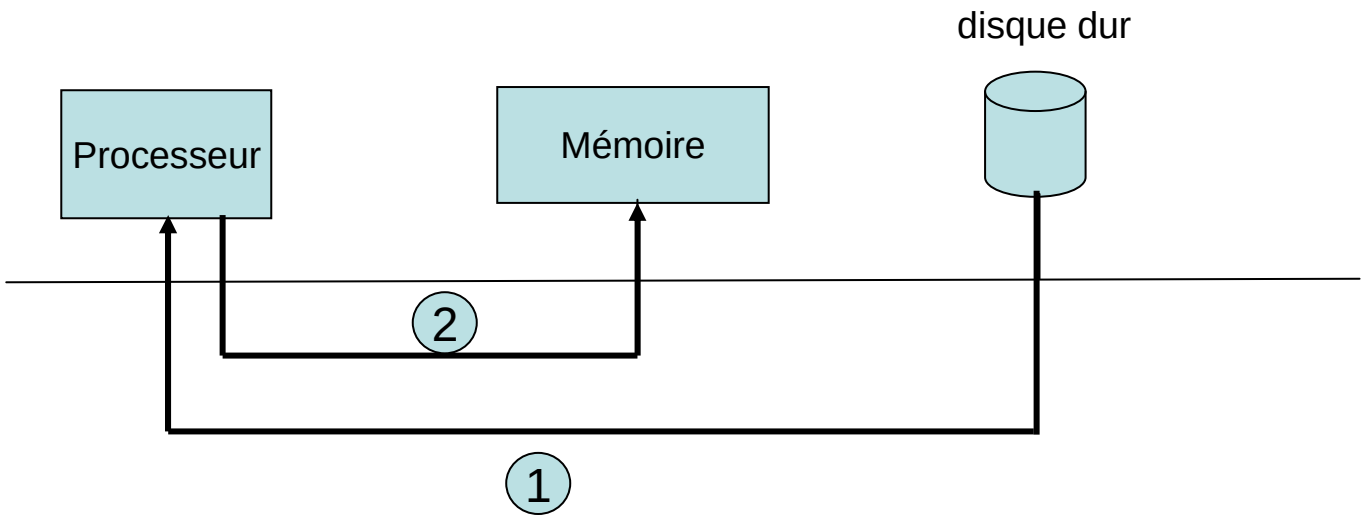
Le résultat est entier, pour calculer sqrt(2), il faut décaler la valeur du registre le plus possible, c'est-à-dire calculer sqrt(2000…000)

Transfert DMA

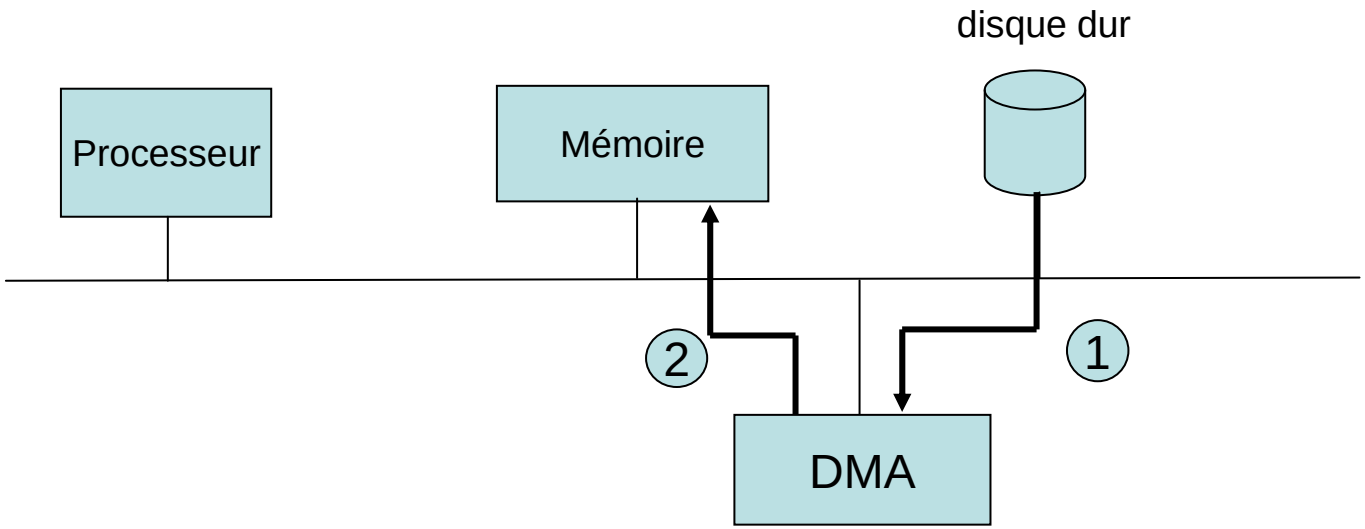
DMA = Direct Memory Access (accès direct à la mémoire)

Le dispositif DMA permet d'effectuer des déplacements de données entre les mémoires ou mémoire <–> unité d'échange sans que le processeur en ait la charge

Transfert sans DMA:



Transfert DMA



GBA - DMA

La GBA contient 4 canaux DMA (DMA0, DMA1, DMA2 et DMA3), le plus prioritaire étant DMA0 puis DMA1, etc.

Le processeur ne peut pas accéder la mémoire pendant qu'un transfert DMA à lieu.

Les registres:

0x40000B0 : DMA0SAD – DMA0 Source address (W) (memoire interne)

0x40000BC : DMA1SAD

0x40000C8 : DMA2SAD

0x40000D4 : DMA3SAD

GBA - DMA

Les registres :

0x40000B4 : DMA0DAD – DMA 0 destination address (mémoire interne)
0x40000C0 : DMA1DAD – (mémoire interne)
0x40000CC : DMA2DAD – (mémoire interne)
0x40000D8 : DMA3DAD – toutes mémoires

0x40000B8 : DMA0CNT_L – DMA 0 Word count (14 bits)
0x40000C4 : DMA1CNT_L – DMA 1
0x40000D0 : DMA2CNT_L – DMA 2
0x40000DC : DMA3CNT_L – DMA 3

DMA - GBA

Les registres :

0x40000BA : DMA0CNT_H – DMA 0 Control (R/W)

0x40000C6 : DMA1CNT_H – DMA 1

0x40000D2 : DMA2CNT_H – DMA 2

0x40000DE : DMA3CNT_H – DMA 3

DMAxCNT_H

Bits	Signification
0 - 4	inutilisés
5 -6	Destination addr. CRTL (0 = incrémente, 1 = décrémente, 2 = Fixed)
7 - 8	Source addr. CRTL (idem)
9	DMA repeat (0=Off, 1 = On)
10	Type de transfert (0 = 16bits, 1=32 bits)
11	spécialisé
12 – 13	Début du transfert (0 = imédiat, 1 = VBlank, 2 = HBlank)
14	Interruption après transfert (0 = non, 1 = oui)
15	DMA Enable

DMAxCNT_H

Bit 9 : Repeat

Si le bit est positionné à 1 le bit 15 (enable) n’est pas mis à zéro après le transfert et un nouveau transfert DMA aura lieu automatiquement dès que la condition de départ sera valide (immédiat, HBlank, VBlank)

Exemple C

```
#define REG_DMA3SAD *(volatile unsigned int*)0x40000D4
#define REG_DMA3DAD *(volatile unsigned int*)0x40000D8
#define REG_DMA3CNT *(volatile unsigned int)0x40000DC
#define REG_ENABLE 0x8000000
#define DATA_TIMING_IMMEDIATE 0x00000000
#define DMA_16 0x00000000
#define DMA_32 0x04000000

#define DMA32NOW  (DMA_ENABLE | DMA_TIMING_IMMEDIATE|
    DMA_32)

#define DMA16NOW  (DMA_ENABLE | DMA_TIMING_IMMEDIATE|
    DMA_16)
```

Exemple C

```
void DMAFastCopy(void * source, void *dest, unsigned int count,
    unsigned int mode)

    if (mode == DMA_16NOW || mode == DMA_32NOW)
    {
        REG_DMA3SAD = (unsigned int)source;
        REG_DMA3DAD = (unsigned int)dest;
        REG_DMA3CNT = count | mode;
    }
```

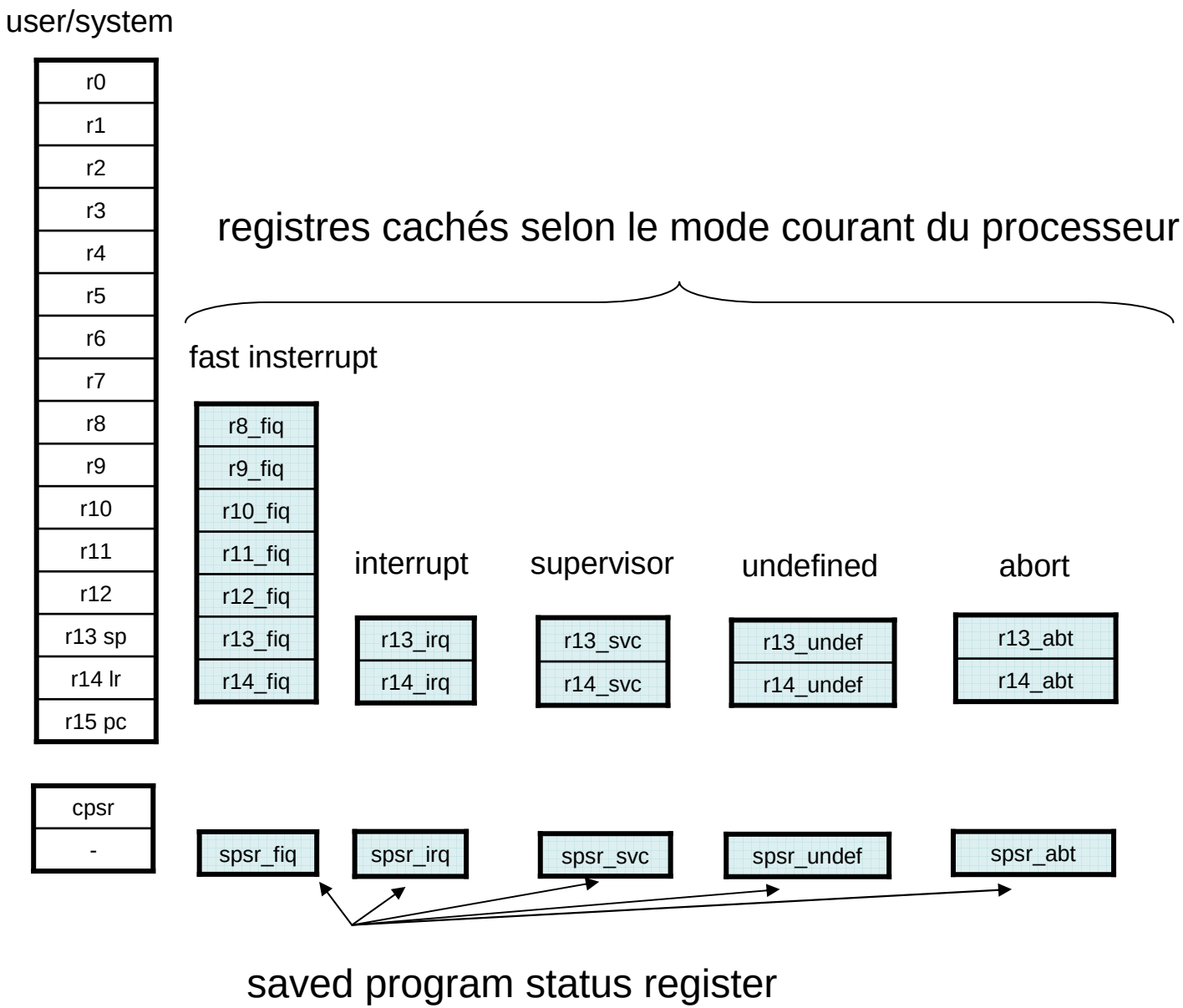
Généralités sur les interruptions

On rappelle que le processeur ARM7TDMI fonctionne dans différents modes :

- fast_interrupt
- interrupt
- supervisor
- undefined
- abort

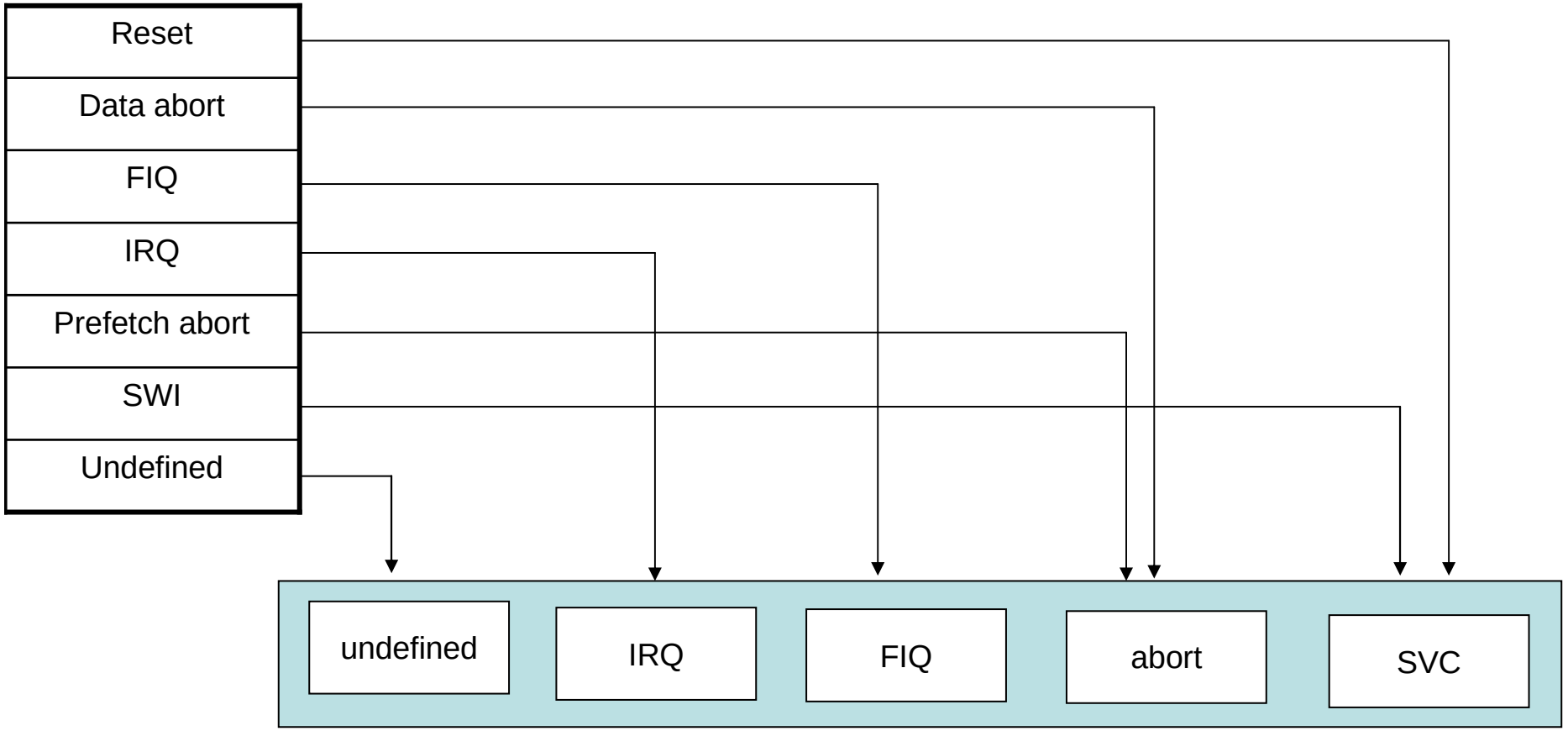
A chacun de ces modes correspond différents registres:

Généralités



Généralités

Les interruptions et les modes associés sont déterminés par la relation suivante



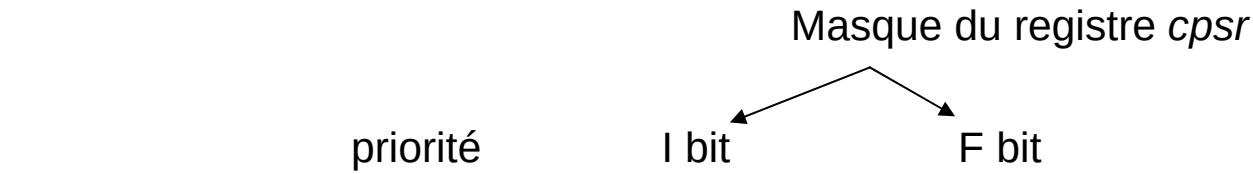
Généralités

Lorsqu'une exceptions est levée, le processeur ARM automatiquement procède à

- La sauvegarde du registre *cpsr* dans le registre *spsr* correspondant au mode de l'exception
- Sauvegarde le *pc* dans le registre *lr* correspondant au mode de l'exception

Priorités

Les différentes sources d'exceptions sont associées à des priorités selon la table suivante



Reset	1	1	1
Data abort	2	1	-
FIQ	3	1	1
IRQ	4	1	-
Prefetch abort	5	1	-
SWI	6	1	-
Undefined	6	1	-

Priorité

L’exception **Reset** est la plus prioritaire, elle est levée lorsque le processeur est mis sous tension. Lorsque cette interruption est levée, elle est prioritaire sur toutes les autres.

Le gestionnaire de l’exception **Reset** s’occupe, d’initialiser les interruptions externes, le décodage des mémoires, …

Il faut éviter de lever une exception lors de l’exécution des premières instructions du gestionnaire de **Reset**, c’est-à-dire ne pas utiliser d’instruction SWI, d’instructions non définie, et d’accès mémoire qui peuvent ne pas aboutir.

Priorités

Lors du traitement d'une exception **Data abort**, une interruption **FIQ** peut être levée. Le gestionnaire de l'exception **Data abort** va perdre la main et la reprendre lorsque le gestionnaire de l'interruption **FIQ** va se terminer.

On remarque que le masque correspondant à **FIQ** est positionné ce qui à pour effet d'inhiber les interruptions **FIQ** dans la routine de traitement. A priori le traitement ne peut pas être interrompu par une autre interruption externe.

FIQ – IRQ - SWI

Pour le programmeur, les trois sources d'exceptions utilisées sont FIQ, IRQ et SWI.

• SWI est généralement utilisée pour procéder à des appels système, en mode privilégié (SVC).

• IRQ est utilisé pour la gestion des interruptions 'générales', c'est-à-dire qui supportent des temps de latence et une plus faible priorité que les exceptions FIQ.

• FIQ est utilisé pour traiter une source d'interruption spécifique, le plus rapidement possible. Par exemple, la gestion des accès directs en mémoire.

FIQ - IRQ

En effet, un gestionnaire d'interruption FIQ possède des copies des regsitres r8-r14. Ces registres peuvent être utilisés pour mémoriser des variables locales, compteurs, pointeurs,…

C'est particulièrement efficace si une seule source d'interruption utilise FIQ.

Un autre avantage est que la première instruction exécutée lorsqu'une interruption FIQ est levée se trouve à la fin de la table d'interruption . Le gestionnaire peut commencer directement sans utiliser d'instruction de branchement.

La GBA n'utilise pas l'interruption FIQ.