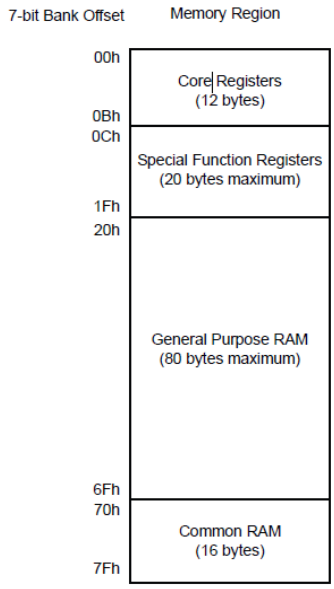
1. RAM

RAM паметта е разделена на 32 банки по 128 байта. Всяка банка има следното устройство:



Основни регистри (Core Registers): Това са регистри които имат непосредствен ефект в/у операциите. Във всичките 32 банки си остават едни и същи:

CORE REGISTERS

Addresses	BANKx
x00h or x80h	INDF0
x01h or x81h	INDF1
x02h or x82h	PCL
x03h or x83h	STATUS
x04h or x84h	FSR0L
x05h or x85h	FSR0H
x06h or x86h	FSR1L
x07h or x87h	FSR1H
x08h or x88h	BSR
x09h or x89h	WREG
x0Ah or x8Ah	PCLATH
x0Bh or x8Bh	INTCON

Специални регистри (Special Function Registers): Регистри с които се управлява периферията на процесора

Универсални регистри(General Purpose RAM): Това са регистри с общо предназначение

Общи универсални регистри (Common RAM) : Това са регистри с общо предназначение които са еднакви за всички банки.

Последната банка 31 е малко по специална. Освен основните и общите универсални регистри, тук се намират регистрите управляващи стека (STKPTR, TOSL и TOSH). Най-важното е че тук се намират т.н. SHADOW регистри, в които се запомнят някои основни регистри при възникване на прекъсване. Регистрите които се запомнят при възникване на прекъсване са:

- W
- STATUS (без TO и PD)

- BSR
- FSR
- PCLATH

След излизането от прекъсване с командата *retfie*, горните регистри се въстановяват автоматично. Ако искаме все пак промяната на даден регистър да се запази и след излизане от прекъсване, просто променения регистър трябва да се запише в съответния SHADOW регистър в банка 31

Bank 31

FA0h FE3h	Unimplemented Read as '0'
FE4h	STATUS_SHAD
FE5h	WREG_SHAD
FE6h	BSR_SHAD
FE7h	PCLATH_SHAD
FE8h	FSR0L_SHAD
FE9h	FSR0H_SHAD
FEAh	FSR1L_SHAD
FEBh	FSR1H_SHAD
FECh	—
FEDh	STKPTR
FEEh	TOSL
FEFh	TOSH

Legend: = Unimplemented data memory locations, read as '0'.

Анка3

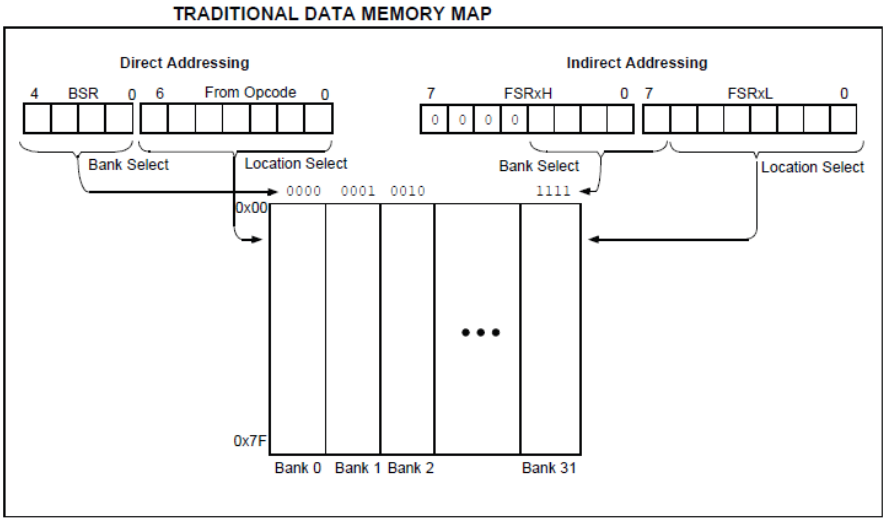
W регистъра е част от RAM, следователно може да участва в операции с регистри, като rrf. Мнемониката му е WREG.

2. Адресиране на RAM

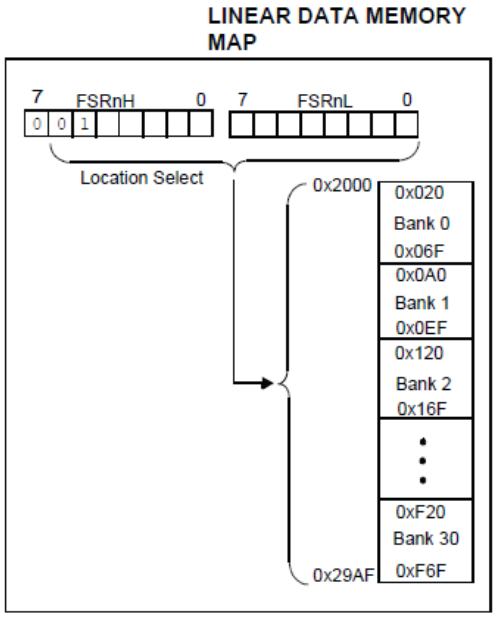
Директната адресация се извършва чрез регистъра BSR, който съдържа банката с която се работи в момента. Зареждането на BSR става с инструкцията **MOVLB k,** където **k** е номера на банката.

Индиректната адресация използва 16 битовите регистри FSRn, състоящи се от FSRnH и FSRnL. С тях може да се адресират три типа памет:

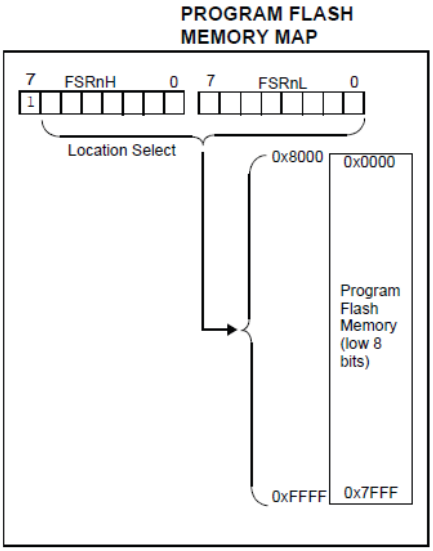
Traditional Data Memory – Това са всъщност 32те банки RAM.Адресацията им е както досега и няма нищо особено.



Linear Data Memory – Това е виртуална памет в която са подредени всички универсални регистри един под друг. Така няма нужда да се прескача от банка в банка. Неудобството е че адресите трябва да се пресмятат, тъй като на адрес 0x2000 отговаря адрес 0x20 от банка 0 а адрес 0x206F ще отиде вече в 0xBF в банка 1. **Common RAM регистрите (x70:x7F) не влизат в Linear Data Memory !!!**



PROGRAM FLASH MEMORY – Така може да се адресира програмната памет (само за четене!) и да се четат таблици.



ACCESSING PROGRAM MEMORY VIA FSR

```
constants
    RETLW DATA0         ;Index0 data
    RETLW DATA1         ;Index1 data
    RETLW DATA2
    RETLW DATA3
my_function
    ;… LOTS OF CODE…
    MOVLW   LOW constants
    MOVWF   FSR1L
    MOVLW   HIGH constants
    MOVWF   FSR1H
    MOVIW   0[FSR1]
;THE PROGRAM MEMORY IS IN W
```

Горният пример не е много подходящ. Инструкциите moviw 0[FSRx] и movf INDFx, W при проба заредиха „0” във W, т.е. не работят!!! Очакваните резултати се получиха при командите ++FSRx, FSRx++, - -FSRx, FSRx- -. Още една особеност, ако зареждате FSR регистъра както в горния пример, т.е. ако с думата constants определяте начален адрес от данни намиращ се във FLASH паметта, то MPLABa автоматично ще вдигне бит 7 на FSRxH

3. Стек

Стека е както при 16F, т.е. с големина16 думи и преобръщащ се. Разликата е бит **STVREN в CONFIGURATION WORD 2.** Ако STVREN=0 то 17 опит за запис в стека ще презапише върху запи1, 2рия в/у запис 2 и т.н. При STVREN=1 при препълване или при опит за четене при празен стек ще предизвика Ресет. Битовете за препълване и за четене при празен стек се вдигат независимо от STVREN.

4. Нови команди в 16F1

ADDFSR FSRn, k →Добавя 6 битова константата **k** към двойката FSRnH:FSRnL.
$-32 \le k \le 31$, $n \in [0, 1]$, FSR(n) + k → FSR(n). Status → не се променя.

ASRF f {,d} →Аритметично преместване надясно
$0 \le f \le 127$, $d \in [0,1]$, (f<7>)→ dest<7>, (f<7:1>) → dest<6:0>, (f<0>) → C. Status →C, Z (При преместването бит7 не се променя. Битове <7:1> се преместват надясно и отиват във битове <6:0>. Бит0 отива във флага C. Обикновенно се използва за делене на 2 на числа със знак)

ADDWFC f {,d}→Събиране с пренос.
$0 \le f \le 127$, $d \in [0,1]$, (W) + (f) + (C) → dest. Status →C, Z, DC

BRA label
BRA $+k → Относителен преход
$-256 \le$ label - PC + 1 ≤ 255, $-256 \le k \le 255$, (PC) + 1 + k → PC, Status → не се променя.
(Разликата с Goto е че при BRA прехода е относителен. Т.е. при BRA PC се събира с

константата. При Goto константата се записва директно в PC. Следователно при goto прехода е винаги в границата на 2Кб(ако не се пипа PCLATH). При BRA границата от 2Кб може да се премине, което според мен може да предизвика объркаване. Затова BRA не я препоръчвам!)

BRW → Относителен преход с W
(PC) + (W) → PC, W се приема за безнакова константа, Status → не се променя.
(Преди се използваше ADDWF PCL. Разликата е че при ADDWF PCL се променя само младшата част на PC, поради което таблицата трябваше да е в една банка. Другото отличие е че BRW се изпълнява за 2 цикъла)

CALLW →Преход към подпрограма със W
(PC) +1 → TOS, (W) → PC<7:0>, (PCLATH<6:0>) → PC<14:8> Status → не се променя.
Първо адреса за връщане се поставя в стека ((PC) +1 → TOS). След това W се прехвърля в младшият байт на PC. Старшият байт на PC се всема от PCLATH.

LSLF f {,d}→Логическо преместване наляво (C←F← 0)
0 ≤ f ≤ 127, d ∈ [0,1], Status →C, Z

LSRF f {,d}→Логическо преместване надясно (0→F→C)
0 ≤ f ≤ 127, d ∈ [0,1], Status →C, Z

MOVIW ++FSRn	→	FSRn=FSRn+1,	F(FSRn) → W
MOVIW –FSRn	→	FSRn=FSRn-1,	F(FSRn) → W
MOVIW FSRn++	→	F(FSRn) → W,	FSRn=FSRn+1
MOVIW FSRn--	→	F(FSRn) → W,	FSRn=FSRn-1
MOVIW k[FSRn]	→	FSRn не се променя,	F(FSRn+k) → W

n ∈ [0,1], -32 ≤ k ≤ 31, Status → Z

MOVLP k → Зареждане на PCLATH
0 ≤ k ≤ 127, k → PCLATH, Status →не се променя

MOVLB k → Зареждане на регистъра BSR
k 0 ≤ k ≤ 31, k → BSR, Status → не се променя.

MOVWI ++FSRn	→	FSRn=FSRn+1,	W → F(FSRn)
MOVWI –FSRn	→	FSRn=FSRn-1,	W → F(FSRn)
MOVWI FSRn++	→	W → F(FSRn),	FSRn=FSRn+1
MOVWI FSRn--	→	W → F(FSRn),	FSRn=FSRn-1
MOVWI k[FSRn]	→	FSRn не се променя,	W → F(FSRn+k)

n ∈ [0,1], -32 ≤ k ≤ 31, Status →не се променя

OPTION → Зареждане на OPTION_REG
(W) → OPTION_REG, Status →не се променя

RESET → Софтуерен ресет
Status →не се променя

SUBWFB f,d → Изваждане със заем
0 ≤ f ≤ 127 d ∈ [0,1], (f) – (W) – (B) → dest, Status →C, DC, Z

TRIS f → Зареждане на TRIS регистъра
5 ≤ f ≤ 7 (W) → TRIS register ‘f’

Начален адрес на EEPROM

Device	Address
Most PIC1X MCUs	0x2100
PIC18 MCUs	0xF00000
PIC16F19XX MCUs	0xF000