PSM Syntax

Each line of your PSM file should adhere to the following basic syntax. Don't worry too much about getting everything perfect or tidy because the assembler will look after things like additional spaces and is very tolerant of upper or lower case characters except where it really matters. If you get something wrong the assembler will show you what it doesn't like and provide advice to help you correct it.

label: instruction operand1, operand2 ; comment

Any line can be given a label that will eventually be a associated with an address. When a label is defined it must be followed by a colon '**:**' . A label is case sensitive and can be any number of the standard characters 'a' to 'z' , 'A' to 'Z', '0' to '9'' and '_' (underscore) but it should not be a name that could be confused with a hex value

A label can then be used anywhere in the program to define the target address for a JUMP or CALL instruction as well as with the 'lower and 'upper attributes to define constants for use in other instructions.

Any of the KCPSM6 instructions or an assembler directive. Upper or lower case accepted.

All instructions and directives except RETURN have at least one operand and this should be separated from the instruction by at least one space.

Instructions and directives that require a second operand should be separated from the first operand by a comma '**,**' (any spaces are formatting).

Anything following a semicolon '**;**' will be treated as a comment and otherwise ignored by the assembler.

Hint - The assembler ignores empty lines so use an empty comment (just a semicolon) to preserve a blank line in the FMT and LOG files.

Lines only containing comments will be formatted in-line with the instructions. Comments on lines containing an instruction will be formatted in a column to the right of the longest instruction. Looks nice ☺

Default register names for use as 'sX' or 'sY'
's0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 'sA', 'sB', 'sC', 'sD', 'sE', 'sF'.
The assembler will accept upper and lower case, e.g. 'sb', 'SB' and 'Sb' are also 'sB'.

Constant values 'aaa', 'kk', 'ss', 'p' and 'pp.
Each character represents the requirement for a hexadecimal digit to define an address, constant or port. So for example 'kk' is any value in the range '00' to 'FF' hex.
Hex values are the default and can be specified in upper or lower case, e.g, '6d' or '6D'.
Decimal and binary values can be defined using 'd and 'b attributes
e.g. 109'd and 01101101'b are both the same value as '6D'.
Also for 'kk' constants only…
The ASCII equivalent of a character can be assigned, e.g. "n" is the same as '6D'
The lower 8-bits of an address can be identified using label**'lower**.
The upper 4-bits of an address can be identified using label**'upper** (msb 4-bits are zero).

Directives.....
CONSTANT name, kk / ADDRESS aaa / NAMEREG oldname, newname / STRING name$, "text" / TABLE name#, [kk,kk,kk,..] / INST hhhhh

PLEASE SEE – 'all_kcpsm6_syntax.psm' which provides a PSM file (albeit not a real program) that further describes all directives and has examples of all the supported syntax. Since it is a valid PSM file you can assemble it to see the FMT and LOG files as well.

Registers and the NAMEREG Directive

KCPSM6 can generally access 16 general purpose registers assigned the default names ‘s0’ through to ‘sF’. There are absolutely no restrictions on which register or combination of registers can be specified as ‘sX’ or ‘sY’ operands in any of the instructions that work with registers. This provides you with complete freedom to allocate registers as you wish. If you are careful with your allocation of registers to different tasks it will often avoid the requirement to ‘shuffle’ data around too much which is often the case when a processor has an accumulator based processor architecture.

The KCPSM6 assembler is able to identify the default name of a resister regardless of the mixture of upper and lower case characters that you use to describe it but it will always convert it to the lower case ‘s’ followed by an upper case hexadecimal digit when writing the FMT and LOG files. For example ‘S4’ will be interpreted as the default register name ‘s4’. Likewise, ‘sd’, ‘Sd’ and ‘SD’ will all be interpreted as default register name ‘sD’. In other words, the assembler allows you to concentrate on writing your code without having to be so precise about syntax and format.

NAMEREG Directive

The NAMEREG directive is an *optional* facility that can help you keep track of what data you expect a particular register to contain. Prior to the NAMEREG directive the resister will have the default name such as ‘sB’. Once renamed only the new name will identify the register and that name is case sensitive exactly as you defined it. Changing the name has no effect on the contents of the register or how it can be used.

```
ADD sB, 42
;
NAMEREG sB, Status
;
;
INPUT Status, flags_port
COMPARE Status, 12
;
NAMEREG Status, speed
;
SUB speed, 01
;
NAMEREG speed, sB
;
LOAD sB, 19
```

Default register name applies before the NAMEREG directive.

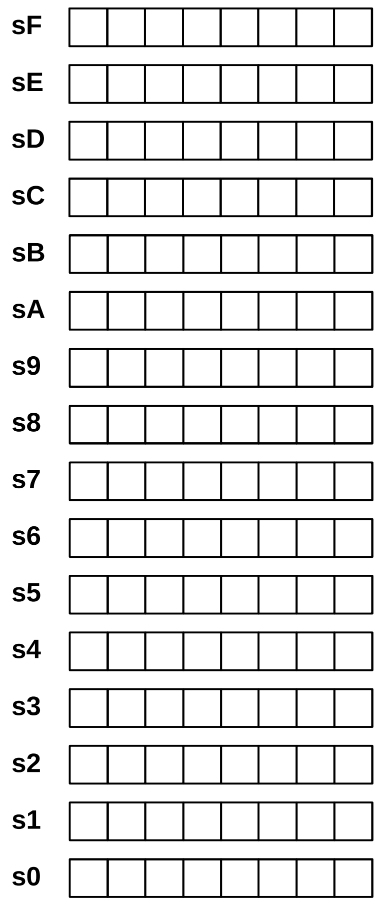
The new name can only contain ‘a’ to ‘z’, ‘A’ to ‘Z’ and ‘_’ underscore (no spaces). It can be any length but must not be a name that could be confused for anything else like a line label of a hexadecimal value.

Following the NAMEREG directive only the new is valid and this name is case sensitive. In this case ‘sB’ will no longer be recognised.

The NAMEREG directive can be used to change the name again and then only the new name is valid in the following code. Depending on your way of thinking this is either useful or something to be avoided! ☺

The appropriate default register name can be restored and following this all the normal case insensitivity rules also apply.

16 Registers
All general purpose
All 8-bits



KCPSM6 Instruction Set

```
aaa : 12-bit address 000 to FFF
 kk : 8-bit constant 00 to FF
 pp : 8-bit port ID 00 to FF
  p : 4-bit port ID 0 to F
 ss : 8-bit scratch pad location 00 to FF
  x : Register within bank s0 to sF
  y : Register within bank s0 to sF
```

Register loading

Page	Opcode	Instruction
55	00xy0	LOAD sX, sY
55	01xkk	LOAD sX, kk
71	16xy0	STAR sX, sY

Logical

Page	Opcode	Instruction
56	02xy0	AND sX, sY
56	03xkk	AND sX, kk
57	04xy0	OR sX, sY
57	05xkk	OR sX, kk
58	06xy0	XOR sX, sY
58	07xkk	XOR sX, kk

Arithmetic

Page	Opcode	Instruction
59	10xy0	ADD sX, sY
59	11xkk	ADD sX, kk
60	12xy0	ADDCY sX, sY
60	13xkk	ADDCY sX, kk
61	18xy0	SUB sX, sY
61	19xkk	SUB sX, kk
62	1Axy0	SUBCY sX, sY
62	1Bxkk	SUBCY sX, kk

Test and Compare

Page	Opcode	Instruction
63	0Cxy0	TEST sX, sY
63	0Dxkk	TEST sX, kk
64	0Exy0	TESTCY sX, sY
64	0Fxkk	TESTCY sX, kk
65	1Cxy0	COMPARE sX, sY
65	1Dxkk	COMPARE sX, kk
66	1Exy0	COMPARECY sX, sY
66	1Fxkk	COMPARECY sX, kk

Shift and Rotate

Page	Opcode	Instruction
67	14x06	SL0 sX
67	14x07	SL1 sX
67	14x04	SLX sX
67	14x00	SLA sX
67	14x02	RL sX
68	14x0E	SR0 sX
68	14x0F	SR1 sX
68	14x0A	SRX sX
68	14x08	SRA sX
68	14x0C	RR sX

Register Bank Selection

Page	Opcode	Instruction
70	37000	REGBANK A
70	37001	REGBANK B

Input and Output

Page	Opcode	Instruction
73	08xy0	INPUT sX, (sY)
73	09xpp	INPUT sX, pp
74	2Cxy0	OUTPUT sX,(sY)
74	2Dxpp	OUTPUT sX, pp
78	2Bkkp	OUTPUTK kk, p

Scratch Pad Memory

(64, 128 or 256 bytes)

Page	Opcode	Instruction
81	2Exy0	STORE sX,(sY)
81	2Fxss	STORE sX, ss
82	0Axy0	FETCH sX, (sY)
82	0Bxss	FETCH sX, ss

Interrupt Handling

Page	Opcode	Instruction
83	28000	DISABLE INTERRUPT
83	28001	ENABLE INTERRUPT
84	29000	RETURNI DISABLE
84	29001	RETURNI ENABLE

Jump

Page	Opcode	Instruction
87	22aaa	JUMP aaa
88	32aaa	JUMP Z, aaa
88	36aaa	JUMP NZ, aaa
88	3Aaaa	JUMP C, aaa
88	3Eaaa	JUMP NC, aaa
89	26xy0	JUMP@ (sX, sY)

Subroutines

Page	Opcode	Instruction
92	20aaa	CALL aaa
93	30aaa	CALL Z, aaa
93	34aaa	CALL NZ, aaa
93	38aaa	CALL C, aaa
93	3Caaa	CALL NC, aaa
94	24xy0	CALL@ (sX, sY)
96	25000	RETURN
97	31000	RETURN Z
97	35000	RETURN NZ
97	39000	RETURN C
97	3D000	RETURN NC
98	21xkk	LOAD&RETURN sX, kk

Version Control

Page	Opcode	Instruction
100	14x80	HWBUILD sX

XILINX®