Cheatsheet MVaP (version 3.2)

Opérations sur la pile

Opcode	Pile	sp	pc	Condition
`PUSHI` *n*	P[sp] := *n*	sp+1	pc+2	
`POP`		sp−1	pc+1	1 ≤ sp
`DUP`	P[sp]:=P[sp−1]	sp+1	pc+1	1 ≤ sp
`ADD` (`SUB`, `MUL`, `DIV`, `MOD`)	P[sp-2]:=P[sp−2] + P[sp−1]	sp−1	pc+1	2 ≤ sp
`SUP` (`SUPEQ`, `INF`, `INFEQ`, `EQUAL`, `NEQ`)	P[sp−2]:= 1 *si* P[sp−2] > P[sp−1], 0 *sinon*	sp−1	pc+1	2 ≤ sp

Données

Opcode	Pile	sp	pc	Condition
`PUSHG` *n*	P[sp] := P[*n*]	sp+1	pc+2	0 ≤ *n* < sp
`PUSHL` *n*	P[sp] := P[fp+*n*]	sp+1	pc+2	0 ≤ fp + *n* < sp
`STOREG` *n*	P[*n*] := P[sp−1]	sp−1	pc+2	1 ≤ sp *et* 0 ≤ *n* < sp
`STOREL` *n*	P[fp+*n*] := P[sp−1]	sp−1	pc+2	1 ≤ sp *et* 0 ≤ fp + *n* < sp
`READ`	P[sp] := *entier lu*	sp+1	pc+1	un entier sur l'entrée standard
`WRITE`		sp	pc+1	1 ≤ sp

Contrôle de flot

Opcode	Pile	sp	pc	fp	Condition
`JUMP` *Label*		sp	**instr**(*Label*)		
`JUMPF` *Label*		sp-1	**instr**(*Label*) *si* P[sp−1]=0, pc+2 *sinon*		1 ≤ sp
`CALL` *Label*	P[sp] := pc+2 , P[sp+1] := fp	sp+2	**instr**(*Label*)	sp+2	
`RETURN`		fp-2	P[fp-2]	P[fp-1]	2 ≤ sp

Note: avant assemblage, les *Label* font référence à des marques `LABEL` *Label* présentes dans le code. Lors de l'assemblage, ils sont remplacés par l'adresse **instr**(*Label*) dans le code.

Flottants

Opcode	Pile	sp	pc	Condition
`PUSHF` *f*	(P[sp],P[sp+1]) := *f*	sp+2	pc+3	
`FADD` (`FSUB`,`FMUL`,`FDIV`)	(P[sp−4],P[sp−3]) := (P[sp−4],P[sp−3]) + (P[sp−2],P[sp−1])	sp−2	pc+1	4 ≤ sp
`FSUP`	P[sp−4]:= 1 *si* (P[sp−4],P[sp−3])	sp−3	pc+1	4 ≤ sp

(FSUPEQ,FINF,FINFEQ,FEQUAL,FNEQ)	> (P[sp−2],P[sp−1]), 0 *sinon*			
READF	(P[sp],P[sp+1]) := *réel lu*	sp+2	pc+1	un nombre flottant sur l'entrée standard
WRITEF		sp	pc+1	2 ≤ sp
ITOF	(P[sp−1],P[sp]):=double(P[sp−1])	sp+1	pc+1	1 ≤ sp
FTOI	P[sp−2]:=int(P[sp−2],P[sp−1])	sp-1	pc+1	2 ≤ sp

Note: Les nombres flottants sont stockés sur deux mots mémoire.

Opérations supplémentaires

Opcode	Pile	sp	pc	Condition
FREE *n*		sp−*n*	pc+2	*n* ≤ sp
ALLOC *n*	P[x] := 0 pour sp ≤ x < sp+*n*	sp+*n*	pc+2	
PUSHR *n*	P[sp−1] := P[P[sp−1] + *n*]	sp	pc+2	1 ≤ sp *et* 0 < P[sp-1] + *n* < sp
STORER *n*	P[P[sp−2] + n] := P[sp−1]	sp−2	pc+2	1 ≤ sp *et* 0 < P[sp-1] + *n* < sp
JUMPR *Label*		sp−1	**instr**(*Label*)+P[sp−1]	1 ≤ sp
PUSHSP	P[sp] := sp	sp+1	pc+1	
PUSHFP	P[sp] := fp	sp+1	pc+1	