גירסה 1.00 - 20.6.2002

Addressing Modes on G-Machine

מסמך זה הורד מהאתר http://underwar.livedns.co.il.
אין להפיץ מסמך זה במדיה כלשהי, ללא אישור מפורש מאת המחבר.
מחבר המסמך איננו אחראי לכל נזק, ישיר או עקיף, שיגרם עקב השימוש במידע המופיע במסמך, וכן לנכונות התוכן של הנושאים המופיעים במסמך. עם זאת, המחבר עשה את מירב המאמצים כדי לספק את המידע המדויק והמלא ביותר.

Nir Adar
Email: underwar@hotmail.com
Home Page: http://underwar.livedns.co.il

מסמך זה הוא חלק מחוברת שעתידה לצאת במהלך קיץ 2002.
הנושאים הנכללים בחוברת הם:

- סקירה קצרה של מערכות ספרתיות.
- שיקולי עלות/יעילות, pipeline.
- זיכרונות סטטיים (SRAM) וזיכרונות דינמיים (DRAM).
- שיטות בקרה ותזמון.
- תקשורת.
- ארכיטקטורת הMAYBE.
- ארכיטקטורת הG-Machine.
- מחשבים סדרתיים.

ידע קודם הנדרש להבנת מסמך זה הוא הבנת ארכיטקטורת הMAYBE.
אנא שלחו הערות ותיקונים אל המחבר.

ה-G-Machine

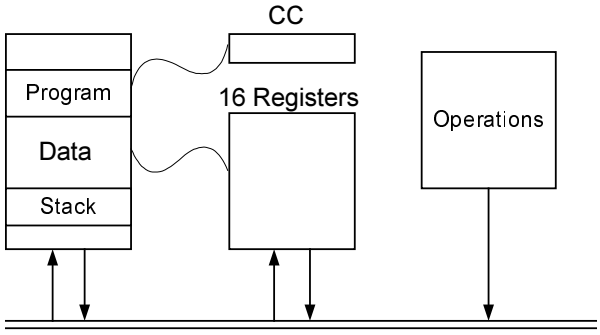
שפת מכונה

נגדיר מכונה וירטואלית שבה אפשר לבצע פקודות יותר מורכבות מאשר בMAYBE.
האינטרפרטר שיפענח פקודות אלו, ייקח אותן ויפרש אותן אחת אחת למיקרו פקודות.
במהלך תכנות בשפה כלשהי, ובפרט בשפת מכונה, מתכנת צריך לעיתים מקום אכסון מהיר, על מנת שיוכל לבצע בו חישובים קריטיים במהירות הגדולה ביותר האפשרית. G-Machine (General Register Machine) היא מכונה המאפשרת למתכנת בשפת המכונה לגשת ולהשתמש ברגיסטרים, שהם התקני זיכרון מהירים.
בחלק זה נכיר את ארכיטקטורה זו, ונראה מימוש של G-Machine על ארכיטקטורת Maybe.

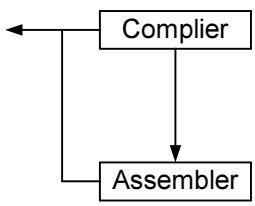
נשים לב לעובדה שעל ידי אינטרפרטרים שונים ניתן לממש מכונות שונות על הMaybe, ולא רק את הG-Machine.
כדי להפריד בין שפת ה μ של הMAYBE לבין שפת המכונה של G-Machine, נוסיף g לפני הפקודות של G-Machine.

מבנה הG-Machine

- 16 רגיסטרים. גודל כל אחד מהם 32bit, ושמותיהם: $GR0,1,2,...,15$.
- זיכרון שגודלו 2^{32} בתים. בזיכרון יושבת התוכנית בשפת מכונה, וכן הנתונים שהתוכנה צריכה על מנת לעבוד.
- הפעולות שמבוצעות ברמה זו הן פעולות כבדות והאינטרפרטר יהפוך כל פעולה לשרשרת פעולות מיקרו.
- האופרנדים יכולים להגיע מהרגיסטרים או מהזיכרון בניגוד לMaybe שם כל האופרנדים מגיעים מהרגיסטרים.



Main Memory	μRom	Control Rom
תוכנית בשפת מכונה + נתונים לתוכנית בשפת מכונה	אינטרפרטר פקודות בשפת מכונה ↓ מיקרו פקודות	מיקרו אינטרפרטר מיקרו פקודות ↓ אותות בקרה (ננו פקודות)



שיטות/אופני מיעון

נרצה לאפשר דרכים שונות לגשת אל האופרנדים בהם אנו משתמשים.
למשל, נרצה להיות מסוגלים לקבל את תוכן רגיסטר כאופרנד, נרצה להיות מסוגלים להתייחס לכתובת בזיכרון, וכו'.
לפיכך, נגדיר שיטות להעביר אופרנדים לפקודות.

האופרנדים יכולים להיות ברגיסטרים או בזיכרון. גישה לרגיסטרים מצריכה 4 ביטים וגישה לזיכרון מצריכה 32 ביטים (רוחב הזיכרון). יש פערים גדולים בצרכים עבור פקודות שמתייחסות אל הזיכרון לבין פקודות שמתייחסות לרגיסטרים.

דרך אפשרית אחת להתייחס אל האופרנדים באופן שונה הוא להוסיף פקודות שונות למכונה, למשל add memory, שתניח כי האופרנד שהיא מקבל הוא כתובת בזיכרון, add register שתניח כי האופרנד שהיא מקבלת הוא רגיסטר וכו'.
החיסרון של פתרון כזה הוא שנצטרך פקודה נפרדת עבור כל צירוף של אופרנדים.

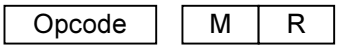
הפתרון שבו נשתמש, הוא הוספת סיביות (M) שתפקידן להגדיר שיטות מיעון (Addressing Modes).

בגישה זו, במקום להעביר לפונקציות את הכתובות של האופרנדים (או את האופרנדים עצמם), בצורה דומה לזו שאנו משתמשים בה בMAYBE, נעביר כתובת כללית (General Address - GA) מהמבנה הבא:

Mode M	Register R	Additional information X (optional)
4 bits	4 bits	(mode dependent)

מבנה זה מאפשר לנו להשתמש לכל היותר ב16 רגיסטרים ולכל היותר ב16 שיטות מיעון.

פקודה תורכב מOpcode באורך שני בתים, ולאחריו האופרנדים השונים, המועברים על ידי GA.

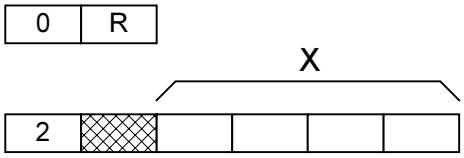


דוגמא

במכונת הG, Mode 0 פירושו שהאופרנד מצוי ברגיסטר, ואילו Mode 2 פירושו כי האופרנד מצוי בזיכרון. נקודד את הGeneral Address כך במקרים הנ"ל:

$M = 0000 \rightarrow$ Operand In Register

$M = 0010 \rightarrow$ Operand In Memory



נניח כי נרצה לבצע את החישוב הבא: $GR4 \leftarrow < GR2 > + < GR3 >$
נשתמש בפקודה gadd4, המבצעת פעולת חיבור על 4 בתים.
בשפת המכונה, תחביר הפקודה יהיה:

gadd4 r(2) r(3) r(4)

r(X) משמעותו שאנו כותבים או קוראים מהרגיסטר X.

הפקודה תקודד בזיכרון באופן הבא:

Byte	Contents	Description
0	00100010	gadd4 opcode
1	00000010	first GA: mode 0 (register), register R2
2	00000011	second GA: mode 0 (register), register R3
3	00000100	third GA: mode 0 (register), register R4

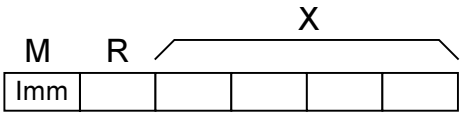
בטבלה הבאה נציג את שיטות מיעון המקובלות במחשבים רבים. לאחר שנסביר שיטות אלו, נציג את קידודן של שיטות המיעון מיעון אלו בG-Machine, ואת השיטות המיוחדות לG-Machine.

Addressing mode	Instruction stream	Assembler syntax	Source datum V	Destination address E
Immediate	X	imm(X)	V=X	
Direct	X	dir(X)	$V=\langle X \rangle$	E=X
Register	R	r(R)	$V=\langle R \rangle$	E=R
Indirect	X	I(X)	$V=\langle\langle X \rangle\rangle$	$E=\langle X \rangle$
Indirect register	R	ir(R)	$V=\langle\langle R \rangle\rangle$	$E=\langle R \rangle$
Indexed	R, X	ix(X, R)	$V=\langle\langle R \rangle+X \rangle$	$E=\langle R \rangle+X$
Stack push		push()		E=SP; $SP \leftarrow \langle SP \rangle+\alpha$
Stack pop		pop()	$SP \leftarrow \langle SP \rangle-\alpha$; $V=\langle\langle SP \rangle\rangle$	
SP-relative	X	ix(X, SP)	$V=\langle\langle SP \rangle+X \rangle$	$E=\langle SP \rangle+X$

הערה

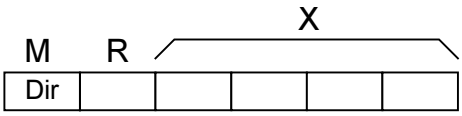
בG-Machine המספרים מאוכסנים בזיכרון בשיטת Little-Endian - הביט הLSB מופיע ראשון והMSB אחרון.

Immediate



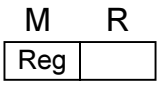
בשיטה זו מתקיים: האופרנד שווה לX.
שיטה זו טובה להגדיר אופרנדים אך לא טובה להגדרת יעד לתוצאת פעולה.
עם זאת, אם ננסה לכתוב ליעד המקודד בשיטת מיעון זו, האינטרפרטר יאפשר לנו לעשות זאת. במקרה כזה המידע ייכתב לתוך X. שיטת פעולה זו איננה מקובלת כיום - בשיטה זו התוכנית משנה את עצמה במהלך הריצה.
מקובל כיום להפריד בין התוכנית לבין הנתונים, ולכן לא נשתמש בשיטה זו בתור יעד לתוצאת פעולה.

Direct



בשיטת מיעון זו, המספר שכתוב בX הוא הEffective Address (EA) של האופרנד - זהו כתובת האופרנד.

Register



בשיטה זו האופרנד הוא רגיסטר. בשדה R נמצא מספר הרגיסטר, ממנו אנו קוראים או כותבים. אין נתונים נוספים (X) בשיטת מיעון זו.

Indirect

בשיטה זו X מכיל את הכתובת של הEffective Address.
מדוע נרצה להשתמש בשיטת מיעון כזו?
שימוש אפשרי הוא כאשר בזמן כתיבת התוכנית לא ניתן לדעת מה תהיה הכתובת של משתנה, ולכן נקצה לכתובת מקום עוד לפני תחילת התוכנית, ונצביע עליו.
שיטת מיעון זו היא הגישה הלא ישירה הפשוטה ביותר הקיימת. שיטות מיעון נוספות, כגון indirect register מכילות בתוכן גם רמה של עקיפות, ומוסיפות עליה דברים נוספים.

Indirect Register

בשיטה זו בתוך הרגיסטר נמצאת הכתובת בזיכרון (EA).

Indexed Addressing

שיטה זו משלבת רמת עקיפות (בעזרת רגיסטר) עם תוספת של קבוע.
הכתובת של האופרנד היא לכן סכום של שני ביטוי: קבוע X השמור עם הפקודה, ומשתנה ברגיסטר R. הכתובת האפקטיבית בשיטה זו היא $E=\langle R\rangle+X$, ואילו התחביר באסמבלי של השיטה הוא ix(X, R).
ישנם שני שימושים עיקריים לשיטה זו:

- פנייה למערכים. במקרה זה אנו מניחים כי המערך מתחיל בכתובת X. ההיסט הרצוי מתחילת המערך מחושב בזמן הריצה ומאוכסן ברגיסטר R. בכל פעם שאנו רוצים לעבור לאיבר הבא, אנו מוסיפים לתוכנו של R את הגודל של איבר בודד, וכך אנו מקבלים בעזרת שיטת מיעון זו את הכתובת של האיבר הבא.

הקוד הבא מממש גישה לאיבר ה-J במערך A:

```
gmove4    dir(J)    r(0)      | R0 ← ⟨J⟩
gmult4    imm(4)    r(0) r(0) | R0 ← ⟨R0⟩·4
gmove4    ix(A, 0)  r(1)      | R1 ← ⟨⟨R0⟩+A⟩
```

לא בהכרח נשתמש בשיטת מיעון זו כדי לגשת למערך.
הקוד הבא מממש גישה לאיבר ה-J במערך שכתובתו מצויה ב-R2:

```
gmove4    dir(J)    r(0)      | R0 ← ⟨J⟩
gmult4    imm(4)    r(0) r(0) | R0 ← ⟨R0⟩·4
gadd4     r(2)      r(0) r(0) | R0 ← ⟨R2⟩+⟨R0⟩
gmove4    ir(0)     r(1)      | R1 ← ⟨⟨R0⟩⟩
```

- גישה למבנים (structs). מבנה הוא אלמנט המוכר לנו משפות עיליות. זהו מעין משתנה, אשר מכיל מספר שדות אשר כל אחד מהם הוא משתנה בפני עצמו. במקרה זה R יכיל את כתובת הבסיס של מבנה כזה, ואילו X יכיל את ההיסט מכתובת הבסיס אל השדה המבוקש.
נביט למשל בביטוי משפת C: p->C, כאשר p הוא מצביע אל מבנה ו-C הוא שם של אחד מהשדות במבנה. במצב זה נוכל להשתמש בשיטת מיעון Indexed Addressing. בקידוד ix(X, R), X יהיה ההיסט של C מהתחלת המבנה, ו-R יהיה כתובת הרגיסטר שיכיל את התוכן של p.

הערה

נשים לב כי אם נשתמש בשיטה זו באופן הבא: ix(0, R), אזי שיטה זו זהה לחלוטין לשיטת indirect register. ניתן לנצל עובדה זו על מכונה המממשת את שתי שיטות מיעון אלו.

מחסנית

נוכל להביא אופרנדים מהמחסנים, ולשמור תוצאות במחסנית.
בניגוד למחסנית של הMAYBE, המחסנית של הG-Machine מתחילה מכתובת נמוכה וגדלה אל הכתובות הגבוהות.

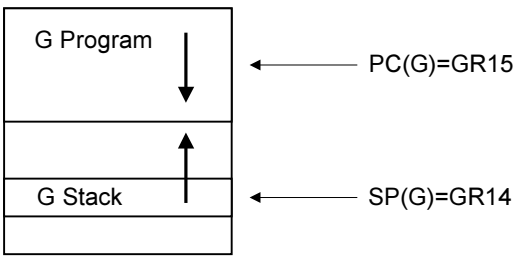
SP-relative addressing

שיטת מיעון זו היא למעשה מקרה מיוחד של indexed addressing, בה הרגיסטר הוא הstack pointer. אנו משתמשים בשיטה זו כדי לגשת לאיבר במחסנית, אך איננו נמצא בראש המחסנית.

G-Machine

עד כה דיברנו על שיטות מיעון כלליות, והדגשנו מעט נקודות הקשורות אליהן ואל הG-Machine. כעת נתרכז במבנה של הG-Machine, ובשיטות המיעון הייחודיות למכונה זו.
G-Machine כוללת מספר תכונות, המאפשרים לה לספק מגוון רחב של מיעונים, בעזרת מספר קטן של מניפולציות:

- בG-Machine הProgram Counter (PC) והStack Pointer (SP) הם רגיסטרים כלליים לכל דבר, ואפשר להשתמש בכל שיטות המיעון עליהם.
  בG-Machine, GR14 משמש כSP ואילו GR15 משמש כPC.
- המכונה כוללת שיטות מיעון עקיפות, אשר, כאפקט צדדי, משנות את התוכן של רגיסטר ספציפי. למשל, שיטות המיעון Post Increment וPre Decrement.
  עובדה זו מאפשרת לנו לסרוק מערך איבר איבר, או ליצור בעזרת רגיסטרים מחסניות נוספות בנוסף למחסנית של המכונה.



שיטות המיעון של G-Machine

הטבלה הבאה מציגה את שיטות המיעון הבסיסיות של ה-G-Machine:

Addressing mode	M field	Effective Address	Assembler syntax
Register	0	$E=Rn$	r(n)
Indirect register	1	$E=\langle Rn\rangle$	ir(n)
Direct	2	$E=X$	dir(X)
Indirect	3	$E=\langle X\rangle$	i(X)
Indexed	4	$E=X+\langle Rn\rangle$	ix(X, n)
Indirect indexed	5	$E=\langle X+\langle Rn\rangle\rangle$	iix(X, n)
Postincrement	6	$E=\langle Rn\rangle; Rn \leftarrow \langle Rn\rangle+\alpha$	posti(n)
Indirect Postincrement	7	$E=\langle\langle Rn\rangle\rangle; Rn \leftarrow \langle Rn\rangle+\alpha$	iposti(n)
Predecrement	8	$Rn \leftarrow \langle Rn\rangle-\alpha;\ E=\langle Rn\rangle$	pred(n)
Indirect Predecrement	9	$Rn \leftarrow \langle Rn\rangle-\alpha;\ E=\langle\langle Rn\rangle\rangle$	ipred(n)

בעזרת שיטות בסיסיות אלו, ובעזרת SP וPC, אנו יוצרים את שיטות המיעון הנוספות.

הטבלה הבאה מציגה את שיטות המיעון הנגזרות משיטות המיעון הבסיסיות:

Addressing mode	M field	General register	Effective Address	Assembler syntax
Immediate	6	PC	$E=\langle PC\rangle; PC \leftarrow \langle PC\rangle+\alpha$	imm$\alpha(X)$
SP-relative	4	SP	$E=X+\langle SP\rangle$	ix(X, SP)
Indirect SP-relative	5	SP	$E=\langle X+\langle SP\rangle\rangle$	iix(X, SP)
Stack (push)	6	SP	$E=\langle SP\rangle; SP \leftarrow \langle SP\rangle+\alpha$	push()
Stack (pop)	8	SP	$SP \leftarrow \langle SP\rangle-\alpha;\ E=\langle SP\rangle$	pop()
PC-relative	4	PC	$E=X+\langle PC\rangle$	rel(X)

כאשר אנו משתמשים בשיטות המיעון הנגזרות משיטות המיעון הבסיסיות, האסמבלר יקודד אותן למעשה לשיטות הבסיסיות, כשרגיסטר הקישור יהיה הPC או הSP.

שיטת המיעון Postincrement

שיטת מיעון זו לא הייתה קיימת בין שיטות המיעון הכלליות שהצגנו. מעצבי ה-G-Machine הוסיפו שיטה זו, עבור פישוט מספר פעולות נפוצות, כגון מעבר על מערכים.
בשיטה זו תוכן הרגיסטר הוא הכתובת האפקטיבית של האופרנד.
לאחר חישוב הכתובת האפקטיבית של האופרנד, מקודם הרגיסטר אל כתובתו של האופרנד הבא.
כתובתו של האופרנד הבא נקבעת לפי הפקודה בה אנו משתמשים. אם נשתמש בפקודה gmove1, למשל, עם שיטת מיעון זו, יקודם ערכו של הרגיסטר ב1. אם נשתמש בפקודה gmove4 לעומת זאת, יקודם ערכו של הרגיסטר ב4.
מכאן, האופרנד יקודם ב1, 2, 4 או 8, בהתאם לפקודות בהן נשתמש.

שיטת המיעון Predecrement

שיטה זו היא שיטת המיעון המשלימה לPostincrement.
בשיטה זו, אנו ראשית מקטינים את ערכו של הרגיסטר, ואז מתייחסים אל תוכן הרגיסטר בתור הכתובת האפקטיבית של האופרנד.

הסיבה ששיטות אלו לא הופיעו ברשימת שיטות המיעון הכלליות, היא שבשיטות אלו אנו מניחים כי לשיטת מיעון יכול להיות Side-effect.
כאשר הגדרנו את שיטות המיעון הכלליות, לא הזכרנו את הטענה הזו כלל וכלל, ולכן לא יכולנו לדבר על שיטות אלו.

הערה

חשוב להדגיש כי אין קשר בין הרגיסטרים הכלליים של ה-G-Machine לבין אלו של ה-Microcode. R0 של ה-G-Machine ממוקם במיקום שונה בSRAM מזה של ה-Microcode. גם בין המחסנית של ה-Microcode לבין המחסנית של G-Machine אין כל קשר, והן אינן תלויות אחת בשניה.
ארבעה בתים בSRAM ישמשו כרגיסטר אחד של ה-G-Machine. בנוסף יוגדרו בSRAM כמה רגיסטרים מיוחדים לחישובי ביניים של ה-G-Machine.

הגדרות המאקרו של שיטות המיעון

נציג כעת את הגדרות המאקרו הממשות את שיטות המיעון שהצגנו.
נדגיש כי הגדרות אלו מיועדת עבור האסמבלי של ה-G-Machine, ובאסמבלי של ה-Microcode לא נעשה בהן שימוש.

להלן ההגדרות של המיעונים השונים:

```
| Addressing-mode macros:

.marco      r(R)          R                 | Register
.marco      ir(R)         0x10+R            | indirect register
.marco      dir(X)        0x20 LONG(X)      | Direct
.marco      i(X)          0x30 LONG(X)      | Indirect
.marco      ix(X, R)      0x40+R LONG(X)    | Indexed
.marco      iix(X, R)     0x50+R LONG(X)    | Indirect indexed
.marco      posti(R)      0x60+R            | Postincrement
.marco      iposti(R)     0x70+R            | Indirect Postincrement
.marco      pred(R)       0x80+R            | Predecrement
.marco      ipred(R)      0x90+R            | Indirect predecrement


| Special "derived" addressing modes:

.marco      imm1(X)       posti(PC) X       | Byte immediate
.marco      imm2(X)       posti(PC) WORD(X) | Word immediate
.marco      imm4(X)       posti(PC) LONG(X) | Long immediate

.marco      rel(X)        ix(X-.-1, PC)     | PC-relative
.marco      push()        posti(SP)         | Stack push
.macro      pop()         pred(SP)          | Stack pop
```

נדגיש מספר נקודות:

- המאקרו WORD(X) מקבל מספר X ופורש אותו על 2 בתים.
- המאקרו LONG(X) מקבל מספר X ופורש אותו על 4 בתים.
- האופרנד שאנו שולחים לפקודות imm1, imm2, imm4 חייב להיות בגודל המצוין בפקודה. אם לא נספק את האופרנד בגודל הדרוש, האסמבלר לא יעיר כלל, אך יקרא את מספר הבתים שמצוין בפקודה. במצב זה התוכנית שנכתוב לא תעבוד. ניתן עקרונית ליצור אסמבלר שיבדוק האם האופרנדים מתאימים לפקודות בצורה אוטומטית, אך האסמבלר אותו אנו לומדים איננו מכיל מנגנון כזה.

EOF