INF225 høsten 2003
Prosjekt del 4: kodegenerering

Thomas Ågotnes

19. november 2003

1 Introduksjon

I denne delen av prosjektet skal C- -parseren fra del 3 utvides til å generere maskinkode. Maskinkoden som skal genereres er ikke for en virkelig mikroprosessor, men for en virtuell maskin kalt *BVM* (*Batu Virtual Machine*) – på samme måte som Java-kompilatorer genererer maskinkode for JVM[1]. Problemet er ganske likt problemet med å generere p-kode; vi må ha en operasjonell beskrivelse av maskinen koden skal kjøre på. En slik beskrivelse følger.

2 Batu og BVM

BVM er en virtuell maskin for språket Batu. Batu og BVM ble laget av Roland Kaufmann for institutt for informatikk i 2000. BVM er implementert som et Java-program med grafisk brukergrensesnitt som kan brukes til å visualisere eksekveringen av et Batu-program.

Et eksempl på et Batu-program er vist i Figur 2 på s. 4.

2.1 Batu

Instruksjonene i Batu tar en eller ingen operand(er). Verdiene som Batu-instruksjoner bruker/lager (f.eks. legger på stack/i variable, se "Kjøretidsmiljø") er *typet*; de er enten heltall eller tabeller. Tabeller blir representert ved hjelp av pekere. Batu har instruksjoner for å allokere tabeller dynamisk som genererer en tabell-peker. Det finnes bare en-dimensjonale tabeller. Batu har instruksjoner for å heltalls-indeksere tabeller, men ikke for peker-aritmetikk. BVM bruker garbage collection for å deallokere minne som er allokert dynamisk.

Hvis instruksjon trenger mer enn en operand, henter den dem fra stacken (se under).

Eksekvering skjer sekvensielt, bortsett fra for eksekveringskontroll-instruksjoner.

2.2 Kjøretidsmiljø

Batu-programmer har tilgjengelig en mengde lokale variable, en mengde globale variable og en stack. BVM har ingen registre.

2.2.1 Variable

Variable blir lagt i tabeller (arrays) i stedet for på stacken. Det er to tabeller; for lokale og globale variable (for forskjellen, se under "Funksjoner og eksekveringskontroll"). Batu har instruksjoner for å lese/skrive variable via indekser til tabellene.

En variabel kan inneholde en heltalls-verdi eller en tabell-peker. Variable er ikke typet, dvs. at den samme variabelen kan inneholde et heltall på ett tidspunkt og en tabell-peker på et annet. Det er forskjellige instruksjoner for å lese en variabel som heltall og som tabell-peker. Forsøk på å lese en variabel som har en heltalls-verdi som en tabell-peker, eller omvendt, resulterer i en kjøretidsfeil.

2.2.2 Stack

Batu har vanlige instruksjoner for å manipulere stacken på en LIFO-måte. (BVM har faktisk flere stacker samtidig, men bare en av dem benyttes om gangen. Se under "Funksjoner og ekseskveringskontroll"). På samme måte som for variable blir heltall lagret som verdier på stacken, mens tabeller blir lagret som pekere. Og, igjen på samme måte som

[1] Hvis vi ser bort fra instruksjonene `read` og `print` er Batu faktisk en undermengde av språket som JVM bruker.

for variable, det finnes forskjellige instruksjoner for å lese hhv. heltall og tabell-pekere fra stacken, og disse er type-sikre.

2.3 Funksjoner og eksekveringskontroll

Et BVM-program er bygget opp som en samling med funksjoner. Hver funksjon er igjen en rad med instruksjoner. Man kan hoppe vilkårlig mellom instruksjoner innad i funksjonen, men ikke mellom funksjoner. Kontroll kan kun overføres til en annen funksjon gjennom et eksplisitt funksjonskall hvor funksjonen som kalles adresseres ved navn i motsetning til adresse. På denne måten kan man generere kode for et funksjonskall uten å slå opp adressen til funksjoner i symboltabellen. BVM lager selv navn for funksjonene i et Batu-program ved å slå sammen identifikatoren vi gir funksjonen med antall og typene til argumentene. Dette kalles et *sammenfoldet navn* og er beskrevet under. Det er sammenfoldede navn man må gi som argument til Batu-instruksjonene som utfører et funksjonskall.

Alle instruksjoner i et Batu-program må forekomme inne i en funksjon. Startpunktet for et program er funksjonen med (sammenfoldet) navn `main()I`.

Ellers angis hopp mellom instruksjoner med den relative avstand mellom avgang- og ankomststed. Et hopplengde på 1 vil føre kontrollen til neste instruksjon. (En hopplengde på 0 vil føre til ikke-terminering). Hopp utenfor funksjonens rekkevidde er udefinert.

2.3.1 Sammenfolding av funksjonsnavn

Et funksjonsnavn i BVM er satt sammen av:

- En identifikatoren, f.eks. `foobar`.
- Sammenfoldede typenavn for alle parametre mellom `(` og `)` (uten komma mellom).
- Returtypen: `V` eller `I`

For de enkle typene, er det sammenfoldede navnet en stor bokstav, «I» for int og «V» for void. Det sammenfoldede navnet på ein tabell er en venstreklamme «[» etterfulgt av størrelsen og grunntypen («I» for int eller rekursivt navnet på en tabelltype). Størrelsen kan sløyfes hvis den ikke finnes (som for funksjonsparametre). Eksempler: se Tabell 1.

2.3.2 Ramme

Når kontroll overføres til en funksjon, eksekveres funksjonen innenfor en egen ramme. En ramme består av en mengde lokale variable og en stack. I tillegg har en funksjon tilgang på globale variable. Når funksjonskallet returnerer forsvinner rammen.

Staren på hver funksjon i et Batu-program deklareres med `.function`, og rammen må spesifiseres vha. deklarasjonene `.locals` som angir antall lokale variable (inkludert parametre) og `.stack` som angir hvor stor stack funksjonen trenger.

Lokale og globale variable Alle funksjonene har tilgang til den samme tabellen med globale variable. I tillegg har hver funksjon tilgang til sin egen tabell av lokale variable, som ikke er tilgjengelige fra andre funksjoner.

Disse tabellene brukes ikke direkte; Batu har egne instruksjoner for å hente ut/sette verdiene til de globale/lokale variablene via en indeks som angir variabel-nummer. Det er kompilator-skriverens ansvar å assosiere variabel-navn (i kildekoden) med variabel-nummer (i Batu-koden); Batu har ingen form for variabel-navn (som for funksjonsnavn).

Lokal stack Hver aktivering av en funksjon har sin egen stack, som er tom ved starten av eksekveringen av funksjonen. Funksjonen har ikke tilgang på stacken til funksjonen som kalte den.

2.3.3 Parameteroverføring

I et funksjonskall vil BVM automatisk flytte argumentene fra den kallende funksjonens stabel inn i den kalte funksjonens lokale variabler. Det siste parameteret ligger øverst på stabelen, det neste siste nest øverst og så videre. Den første parameteret blir flyttet inn i den lokale variabelen med indeks 0, det andre parameteret blir flyttet inn i den med indeks 1 og så videre.

Funksjonen har fri tilgang til argumentene, fordi de faktisk er lokale variabler. Men når funksjonen returnerer vil ikke endringer som er gjort i disse være reflektert i den kallende funksjonen, fordi argumentene var verdier som ble hentet fra stabelen.

Deklarasjon	Sammefoldet navn
`int main(void) {..}`	`main()I`
`void foo(int i, int j) {..}`	`foo(II)V`
`int bar(int i[], int j, int k[]) {..}`	`bar([II[I)I`

Tabell 1: Eksempler på sammenfolding av funksjonsnavn

(Verdier i en tabell vil være endret, men det er fordi disse ble endret indirekte. Endringer i referansen til tabellen vil ikke være reflektert).

Ved retur flyttes en returverdi fra den kalte funksjonens stabel tilbake til den kallende funksjonens stabel. De opprinnelige argumentene til funksjonen er bort fra den kallende funksjonens stabel. Ved retur gjenopptar den kallende funksjonen alltid kontrollen i instruksjonen som følger funksjonskallet.

2.4 Programfilen

Programmer som skal kjøre i BVM må være lagret i en bestemt form.

Et BVM-program er i praksis en linjeinndelt fil som overholder følgende punkter. Det er ikke anledning til å putte inn blanke tegn med mindre så er angitt.

- Hver funksjon er innledet med tre deklarasjoner, f.eks.:

  ```
  .function foobar(I)V
  .locals 2
  .stack 3
  ```

  Et direktiv kommer på starten av en linje. Det skal være ett mellomrom mellom direktivet og argumentet.

- Etter direktivene kommer instruksjonene som utgjør funksjonens kropp. Hver funksjon varer til en ny funksjon blir definert eller til slutten av filen.

- En instruksjon er på formen

  ```
  \tinstruksjonskode\ operand\n
  ```

  Merk at bare en blank er tillatt mellom instruksjonskoden og operanden. Dersom instruksjonen ikke tar noen operand, skal det ikke være noen blanke før linjeskiftet.

- Står det en asterisk i forkant av instruksjonen, settes der et stopppunkt (breakpoint) ved denne instruksjonen.

- Instruksjonskodene skal stå i kun små bokstaver.

- Kildeinformasjon har formen:

  ```
  ;tekst\n
  ```

  Slik informasjon kan være nyttig å legge sammen med koden for avlusningsformål.

- Kildeinformasjon teller ikke som instruksjoner.

2.5 Ressurser

Dokumentasjon av instruksjonssettet til Batu finnes på Studentportalen. Der kan også BVM lastes ned som en `.jar`-fil. Hvis du har java installert på maskinen, kan du kjøre BVM med

```
java -jar bvm.jar
```

3 Eksempel

Figur 2 viser mulig Batu-kode for C- -programmet i Figur 1. Merk at mens C- har funksjonen `void main(void)` som startpunkt, har Batu `main()I` som startpunkt. Du må derfor generere kode som returnerer en vilkårlig verdi fra `main()I`.

4 Oppgave

4.1 Innlevering

Besvarelsen skal leveres på e-post til `peter.guzikowski@student.uib.no` innen mandag 1. desember kl. 00:00. Hver student skal levere en selvstendig besvarelse.

```
int foo(int i) {
  int a;
  int b;
  return i;
}
void main(void) {
  int a[10];
  foo(a[0]);
}
```

Figur 1: Eksempel på C- -program

```
;foo(i) {
.function foo(I)I
.locals 3
.stack 1
;  int a, b;
  iconst_0
  istore 1
  iconst_0
  istore 2
;  return i;
  iload 0
  ireturn
;}
  iconst_0
  ireturn
;main() {
.function main()I
.locals 1
.stack 2
;  int a[10];
  ldc_w 10
  newarray
  astore 0
;  foo(a[0]);
  aload 0
  ldc_w 0
  iaload
  invokestatic foo(I)I
  pop
;}
  iconst_0
  ireturn
```

Figur 2: Eksempel på Batu-program

4.2 Innhold

Du skal lage et program som leser C- -kode fra standard-inn og skriver Batu-kode til standard-ut.

Dersom du får problemer med å bli ferdig med alt: det er viktigere å ha gjort noe skikkelig enn å ha begynt på absolutt alt. Start med å lage små C- -programmer som bare bruker enkelte programkonstruksjoner, og få kompilatoren til å virke for disse før du går videre.

Ta i det minste en aritmetisk operasjon (e.g. +), en logisk operasjon (e.g. <), tilordning, if- og while-sats, funksjonskall med ett parameter. Dokumenter hvilke innskrenkninger du gjør. Legg ved et kjøreeksempel som bare bruker det du har implementert.

Det er ikke nødvendig å generere kode dersom det er en feil i C- -programmet.

Besvarelsen må inneholde:

- Fullt dokumentert kildekode (inkludert de tidligere delene av prosjektet).
- En jar-fil med alle klassene som trengs for å kjøre programmet.
- En beskrivelse av programmet i en separat tekst-fil.
- Eksempler på programmer som programmet ditt kan kompilere.

4.3 Hint

- For å lage rammen til en funksjon:
  - Alle parametre må tildeles en (lokal) variabel-indeks
  - Alle lokale variable i en funksjon må tildeles en variabel-indeks
  - Antall lokale variable må beregnes
  - Maksimal stackdybde må beregnes
- Variable må initieres før de kan brukes; dette er tryggest å gjøre rett etter at variablene er deklarert.
- Husk at parametre skal telle med i antall lokale variabler en funksjon bruker (men de skal selvsagt ikke initieres)!

- Dere kan legge inn `output(I)V` og `input()I` i symboltabellen før analysen starter. I produksjonen som gjør funksjonskall, kan dere teste på om det er print eller read som kalles og gi tilbake de riktige kodene for disse i såfall. Ellers genererer dere en `invokestatic`-instruksjon som benytter seg av identifikatoren.

- Det finnes ingen instruksjoner for sammenligningsoperatorer; dere må oversette disse til en liten if-sats hvor dere lar det være igjen 0 (false) eller 1 (true) alt ettersom betingelsen stemte. Husk at resultatet fra sammenligninger også kan tildeles til variabler.

- Dersom alle noder i analysetreet returnerer en vektor med koden sin, så vil størrelsen på denne fortelle hvor langt hopplengde skal være for å hoppe over den.

- Ikke skriv kode rett ut, men saml instruksjonene opp i en Vector for hver produksjon. Hver produksjon bygger sammen sin egen Vector ved hjelp av koden fra barna sine samt litt kode for å limedisse sammen.

- Der finnes ingen instruksjon for logisk-IKKE; men istedenfor not x kan dere benytte dere av mønsteret const_1, ixor.

- Å finne maksimal stabeldybde er en klassisk DFS-algoritme. Stabeldybden vil være av samme orden som analysetreet.

- Uttrykk skal etterlate verdien sin på stabelen. Siden tilordning også er et uttrykk, må dere bruke dup (eller dup_x2 for tabelltilordninger) før dere skriver verdien til minnet, siden instruksjonene i/a/iastore fjerner verdien som den lagrer.

OBS! Start med oppgaven tidlig, ikke vent til at alle emner blir gjennomgått på forelesning.

Lykke til.