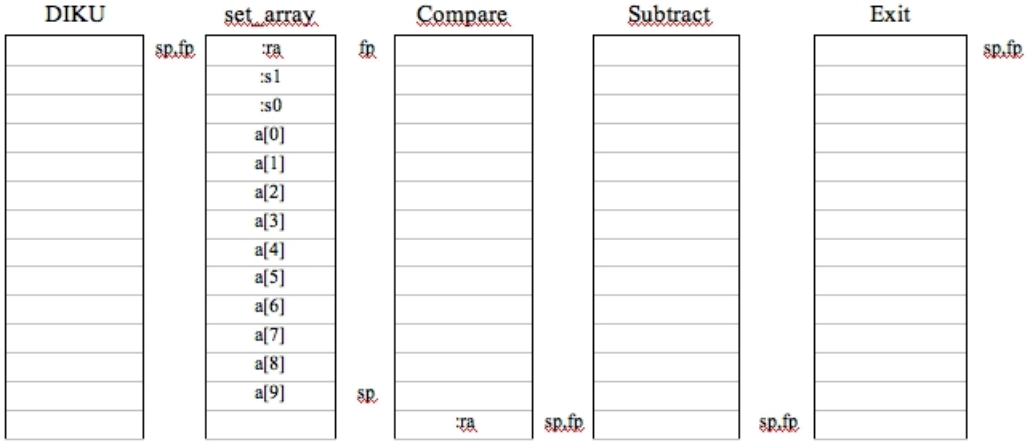
Maskinarkitektur, G2

Daniel Bejder, Anders Bjerg Pedersen
Hold 2

29. november 2006

Gennemgang af kildekoden

Programmet starter med kodeafsnittet med label `DIKU`, hvor pointerne *:sp* og *:fp* initialiseres. Herefter starter selve programmet `set_array`, som indeholder underafsnittet `forLoop`. I `set_array` gør vi plads på stakken og gemmer vores return address *:ra* og vores saved registers *:s1* og *:s0*. Derefter sættes *:s0* til at pege på første element i arrayet ($a[0]$). `set_array` starter med parameteren `num` gemt i register *:a0*.
I udførelsen af `forLoop` gemmer vi framepointeren for `set_array` og gemmer `i` i *:a1*, tjekker om arrayindekset er større end 9 i hvilket tilfælde vi hopper til `Exit`; ellers hopper vi via jump-and-link til `Compare`. Når vi returnerer fra `Compare`, restorer vi vores frame pointer og beregner adressen i stakken på det element, vi er i gang med at se på, og gemmer dette, incrementer `i` og hopper tilbage til `forLoop`.
I `Compare`-afsnittet sætter vi vores lokale frame pointer og gør plads på stakken til og gemmer returadressen *:ra*. Her bruger igen parameteren `num` gemt i register *:a0* og desuden parameteren `i` gemt i register *:a1* (disse variable kaldes nu `a` og `b` i kildekoden til C-programmet). Via jump-and-link hopper vi til `Subtract` (med samme parametre, der derfor ikke gemmes eller ændres), og når vi kommer tilbage, gemmer vi 1 i *v0*, hvis returværdien fra `Subtract` ($v0{=}a-b$) er større end eller lig nul. Vi restorer nu vores return address *:ra*, frigører pladsen og hopper tilbage til `forLoop`.
`Subtract`-afsnittet beregner $a-b$ og bruger jump register *:ra* til at hoppe tilbage til `Compare`.
I `Exit`-afsnittet restorer vi vores saved registers *s0* og *s1* og vores return address *:ra*, resetter stack- og frame pointer til deres oprindelige værdier og stopper.



Figur 1: Oversigt over programmets stack

Overvejelser ifm. valg af operationer

I forbindelse med brugen af stakken følger vi bogens konventioner om, at rækkefølgen, vi gemmer registre i (oppefra og ned), er: argumentregistre, return address, saved registers, lokale arrays og strukturer.

Test og "Brugervejledning"

Når programmet køres (mips G2Final.hex 0) vil det vente på input af `num`, hvorefter det kører færdigt og undervejs udskriver indeholdet af `array`, der er 1 hvis `num` er større eller lig med `i`. Kørsel med num = -1, 0, 3 og 11 giver de forventede 0, 1, 4 og 10 1-taller i array'et, så vi vil mene, at programmet virker, som det skal.